Tampereen yliopisto

Anna-Kaisa Mertanen

KATSAUS NOSQL-TIETOKANTOJEN SUORITUSKYKYYN

Informaatioteknologian ja viestinnän tiedekunta
Pro gradu -tutkielma
Toukokuu 2019

TIIVISTELMÄ

Anna-Kaisa Mertanen: Katsaus NoSQL-tietokantojen suorituskykyyn
Pro gradu -tutkielma, 49 sivua
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Toukokuu 2019

Tässä tutkielmassa käsitellään NoSQL-tietokantojen suorituskykyä. Työssä esitellään NoSQL-tietomallien keskeisimmät ominaisuudet ja vertaillaan NoSQL- ja relaatiotietomallien välisiä eroja. Lisäksi käsitellään tietokantojen hajautusmekanismeja sekä hajautettujen tietokantojen ominaisuuksia määrittelevää CAP-teoreemaa. Työssä perehdytään suorituskyvyn mittaamiseen ja sen mittareihin. Mittaustyökaluista esitellään tarkemmin avain-arvoparitietokantojen vertailuun kehitetty Yahoo! Cloud Serving Benchmark (YCSB). Työn tavoitteena on NoSQL-tietomallien ja niihin perustuvien tietokantojen suorituskyvyn vertailu aihetta käsittelevien tutkimusten, julkaisujen ja artikkeleiden pohjalta. Tutkimuksen tuloksena saatiin tietoa NoSQL-tietomallien ja -tietokantojen suorituskyvystä sekä suorituskykyyn vaikuttavista tekijöistä. Tutkimustuloksia voidaan käyttää apuna suorituskyvyltään käyttökohteeseensa parhaan mahdollisen tietokannan valinnassa.

Avainsanat: tietomalli, tietokanta, relaatiotietokanta, SQL, NoSQL, suorituskyky, YCSB

Sisällys

1 Johdanto

Tietoa voidaan tallentaa sekä analogisessa että digitaalisessa muodossa. Vielä vuonna 2000 suurin osa maailmassa syntyvästä tiedosta oli analogista: paperia, filmiä ja ääni- ja kuvanauhoja [Dahlberg 2015]. Digitaalinen tarkoittaa numeroihin perustuvaa tai numeroita vastaavia signaaleja käyttävää [Kielitoimiston sanakirja 2018]. Vuonna 2011 kaikesta syntyvästä tiedosta 99 % oli digitaalista [Hilbert and Lopez 2011]. Digitaalisen tiedon määrä on tällä hetkellä suurempi kuin koskaan, arviolta 33 tsettatavua (ZB)[1] ja määrän arvioidaan kasvavan 175 tsettatavuun vuoteen 2025 mennessä [Reinsel *et al.* 2018]. Syitä tietomäärän kasvuun ovat esimerkiksi web-teknologian kehitys sekä verkkoon kytkettyjen laitteiden, kuten mobiililaitteiden ja esineiden internetin (Internet of Things, IoT) sensorien määrän voimakas kasvu [Grolinger *et al.* 2013].

Tämän tietomäärän tallentamiseen, käsittelyyn ja analysointiin tarvitaan valtava määrä tallennustilaa ja uudenlaisia tallennusratkaisuja. Tiedonhallinta onkin siirtynyt yhä enemmän perinteisiltä työasemilta ja yritysten palvelimilta pilvipalveluihin [Bazar and Iosif 2014]. Nykyaikana tuotettava tieto on myös aiempaa monimuotoisempaa. Tieto voi olla rakenteista, rakenteetonta tai puolirakenteista. Rakenteetonta dataa on esimerkiksi vapaasta tekstistä koostuva dokumentti [Elmasri and Navathe 2004].

Tietokanta on kokoelma jollain tavalla toisiinsa liittyvää tietoa [Elmasri and Navathe 2004]. Tietomalli on tapa esittää abstraktio tietokannasta piilottamalla sen yksityiskohdat [Elmasri and Navathe 1989]. Yleisin nykyään käytössä oleva tietomalli on relaatiotietomalli, joka on kehitetty jo 1970-luvulla rakenteisen tiedon hallintaan [Codd 1970]. Relaatiotietomalliin perustuvat relaatiotietokannat ovat luotettavaksi havaittuja monissa eri käyttökohteissa, kuten pankki- ja yritysjärjestelmissä. Nykyisten valtavien tieto- ja käyttäjämäärien ja rakenteeltaan vaihtelevan muotoisen tiedon käsittely tuottaa niille kuitenkin ongelmia sekä suorituskyvyn että tietokantojen skaalautuvuuden suhteen.

Haasteisiin vastaamaan on kehitetty erilaisia, relaatiotietomallista poikkeavia NoSQL (*Not-only-SQL*) -tietomalleja [Grolinger *et al.* 2013]. Suuret verkkopalveluja tarjoavat yritykset Google, Amazon, Facebook ja LinkedIn kehittivät ensimmäisinä omiin tarpeisiinsa sopivat NoSQL-ratkaisunsa, joiden pohjalta on kehitetty lukuisia erilaisia tietokantoja [Bazar and Iosif 2014]. NoSQL-tietomallit jaetaan yleensä neljään pääryhmään: avain-arvopari-, sarakeperhe-, dokumentti- ja graafitietomalleihin. Erilaisista tietomalleista huolimatta voidaan niille yhteisinä, tyypillisinä piirteinä pitää hyvää skaalautuvuutta, saatavuutta ja joustavuutta tiedon tyypin ja rakenteen suhteen. Myös uudet relaatiotietokannat eli NewSQL-tietokannat [Grolinger *et al.* 2013] on kehitetty suurten tietomäärien hallintaan. Niissä yhdistyvät relaaatiotietomallin piirteet NoSQL-tietokannoille tyypilliseen hyvään horisontaaliseen skaalautuvuuteen.

[1] Tsettatavu (ZB) on arvoltaan 10^{21}

Käyttötarkoitukseen epäsopiva tietokanta saattaa aiheuttaa ongelmia suorituskyvyn suhteen, joten oikeanlaisen tietokannan valinta on tärkeää. Tietokantojen suorituskyvyn testaus ja vertailu on kuitenkin vaikeaa, hidasta ja kallista. Erilaisia NoSQL-tietokantoja on tällä hetkellä yli 225 [Edlich 2019]. Tämän työn tavoitteena on tutkia NoSQL-tietomallien ja -tietokantojen suorituskykyä sekä löytää suorituskykyyn vaikuttavia tekijöitä ja näin ollen helpottaa tietokannan valintaa. Työssä on kartoitettu NoSQL-tietokantojen suorituskykyä vertailevia tieteellisiä tutkimuksia ja artikkeleja. NoSQL-tietokantojen suorituskykyä on tutkittu melko paljon. Osassa tutkimuksista on perehdytty relaatio- ja NoSQL-tietokantojen välisiin suorituskykyeroihin ja osassa taas käsitellään pelkästään NoSQL-tietokantojen välisiä eroja. Kokonaiskuvaa näiden perusteella on kuitenkin vaikea hahmottaa. Tutkimusaineistona on yhdeksän erilaista tutkimusta vuosilta 2010 – 2016. Käsiteltäviksi tutkimuksiksi valittiin sellaiset, joissa vertailtiin keskenään erityyppisiä avain-arvopari, dokumentti- ja sarakeperhetietokantoja. Koska graafitietokantojen toimintaperiaate poikkeaa muista, ei niitä käsitteleviä tutkimuksia otettu mukaan tutkimukseen. Tutkimuksen tuloksena löydettiin tietoa tietomallin ja useiden muiden tekijöiden vaikutuksesta tietokannan suorituskykyyn erilaisilla kuormituksilla luku- ja kirjoitusoperaatioita.

Tutkimuksen toisessa luvussa tarkastellaan tietokannan ja tietomallin käsitettä. Luvussa kolme käsitellään relaatiotietomallia ja luvussa neljä esitellään NoSQL-tietomallit. Viidennessä luvussa perehdytään tietokantojen hajautukseen. Luvussa kuusi käsitellään suorituskyvyn mittaamista ja seitsemännessä luvussa käydään läpi suorituskykyä koskeva tutkimusaineisto. Luku on jaettu kahteen osaan: kokeellisiin vertailututkimuksiin sekä tapaustutkimuksiin. Luvusta kahdeksan löytyvät tulokset pohdintoineen. Luvussa yhdeksän ovat tutkimuksen johtopäätökset.

2 Tietokanta ja tietomalli

Tietokanta kuvaa jotakin reaalimaailman kohdetta, jonka kanssa se on jollain tasolla vuorovaikutuksessa. Tietokannan tietokokoelmalla tulee olla luonnollinen merkitys ja sen tulee olla loogisesti yhdenmukainen. Näin ollen satunnaista lajitelmaa tietoa ei voida oikeellisesti kutsua tietokannaksi. Tietokanta on lisäksi luotu tiettyä käyttötarkoitusta, käyttäjäryhmää ja sovellusta varten. Tietokantaa hallitaan tietokannanhallintajärjestelmän (DBMS*, Database Management System*) avulla. Se on kokoelma ohjelmia tietokannan luontia ja ylläpitoa varten. Tietokannanhallintajärjestelmän tehtävät ovat tietokannan määrittely, rakentaminen, manipulointi ja jakaminen. Lisäksi se vastaa muun muassa virhetilanteista toipumisesta ja salauksesta. Tietokannalle voidaan määritellä käytettävä tietotyyppi ja rakenteet sekä tallennettavaa tietoa koskevat rajoitukset. Tietokanta rakennetaan jollekin fyysiselle tallennusvälineelle. Tietokantaa manipuloidaan tietokantakyselyiden avulla ja siitä generoidaan raportteja. Tiedon jakaminen eli samanaikaisuuden hallinta mahdollistaa useiden käyttäjien tai ohjelmien samanaikaisen pääsyn tietokantaan. Tietokanta ja tietokannanhallintajärjestelmä muodostavat yhdessä tietokantajärjestelmän. [Elmasri and Navathe 2004]

Tietokantoja manipuloidaan tietokantaoperaatioiden avulla. Neljä perusoperaatiota ovat luonti, luku, päivitys ja poisto eli CRUD (*Create, Read, Update, Delete*) -operaatiot. Käytettävät operaatiot riippuvat käyttökohteen tarpeista. Lukupainotteiset operaatiot ovat tyypillisiä tiedon haussa, esimerkiksi sensoridatan lukemisessa, tiedon louhinnassa ja analytiikkasovelluksissa. Päivityspainotteista taas on esimerkiksi käyttäjän tekemien toimintojen tallentaminen istunnon aikana tai sensorilukemien kirjoitus tietokantaan. Sovellukset voivat olla joko luku- tai päivityspainotteisia tai kuormitus voi koostua puoliksi molemmista operaatiotyypeistä.

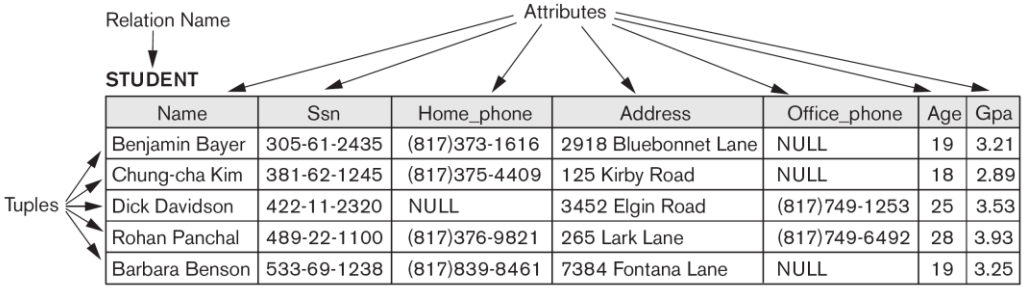
Tietomalli on joukko käsitteitä, joiden avulla voidaan kuvata tietokannan rakenne, tietotyypit, suhteet ja tietoa koskevat rajoitukset [Elmasri and Navathe 1989]. Ullman [1988] määrittelee tietomallin matemaattiseksi formalismiksi, joka koostuu tietoa kuvaavasta merkintätavasta eli notaatiosta ja joukosta operaatioita tämän tiedon manipulointiin. Tietomalli voi olla käsitteellinen, looginen tai fyysinen [Elmasri and Navathe 2004]. Käsitteellisessä tietomallissa tietokanta on kuvattu yleisellä tasolla. Sen avulla määritellään yleisellä, toteutustavasta riippumattomalla tasolla esimerkiksi tietokantaan tulevat kohteet eli entiteetit, niiden ominaisuudet eli attribuutit sekä entiteettien väliset suhteet [Elmasri and Navathe 2004]. Looginen tietomalli suunnitellaan käsitteellisen tietomallin pohjalta. Se kuvaa tiedon rakennetta eli tapaa, jolla tietoyksiköt järjestetään tietokantaan sekä tietoyksiköiden välisiä yhteyksiä. Tietokannan rakenteen loogista kuvausta kutsutaan tietokantakaavioksi eli skeemaksi (*database schema*) [Elmasri and Navathe 1989].

Fyysinen tietomalli kuvaa tietokannan laitteistoarkkitehtuuritasoa, kuten tiedostojen tallennustapaa ja indeksointia [Elmasri and Navathe 1989]. Tietokantojen mallinnuksessa edetään käsitteellisestä mallista loogiseen ja fyysiseen tietomalliin.

Ensimmäiset loogiset tietomallit olivat hierarkkinen tietomalli ja verkkotietomalli. Näiden avulla kuvattiin lähinnä tietokantojen fyysistä toteutustapaa tietokoneessa. Hierarkkinen tietomalli soveltuu tietokannoille, joissa on entiteettien välisiä hierarkkisia suhteita ja luokitteluja. Sen sijaan ei-hierarkkisten suhteiden kuvaaminen on siinä hankalampaa. Vuonna 1964 esitellyssä verkkotietomallissa on olennaista juuri tällaisten suhteiden kuvaaminen. Verkkotietomallin tärkeimmät rakenteet ovat tietue, joukko ja näiden tyypit. Tietueet ovat entiteettejä, jotka koostuvat joukosta toisiinsa liittyviä arvoja. Tietuetyypillä kuvataan tämän tietuejoukon rakennetta. Jokaiselle tietuetyypille ja tietoyksikölle eli attribuutille annetaan nimi ja tyyppi. Tietuetyyppi voi olla esimerkiksi opiskelija ja attribuutteja voivat olla esimerkiksi nimi ja osoite. Verkkotietomallissa voidaan esittää useampia erilaisia suhteita, mikä mahdollistaa monimutkaisemmatkin rakenteet. [Elmasri and Navathe 1989].

3 Relaatiotietomalli

Relaatiotietomallin on kehittänyt IBM:n tutkimuskeskuksen matemaatikko Edgar Frank Codd [Codd 1970]. Relaatiotietomalli on suunniteltu rakenteisen tiedon hallintaan. Rakenteisella tiedolla on ennalta määritelty tietokantakaavio. Relaatio on joukko, joka koostuu monikoista eli tupleista. Relaatiot esitetään tauluina, joissa rivi vastaa yhtä monikkoa. Taulukon sarakkeet ovat attribuutteja. Rivit yksilöidään attribuuttijoukosta koostuvan avaimen avulla. Avaimia voi olla useita, mutta tyypillisesti yksi avain valitaan pääavaimeksi. Relaatioiden väliset viittaukset tehdään viiteavainten avulla. Attribuuteilla on ennalta määrätty tietotyyppi ja arvoalue, joita kaikki arvot noudattavat [Elmasri and Navathe 1989]. Esimerkki relaatiosta on esitetty kuvassa 1. Siinä Student -nimiseen tauluun on tallennettu opiskelijoita koskevaa tietoa. Yhden opiskelijan tiedot ovat yksi monikko yhdellä rivillä. Esimerkiksi nimi, puhelinnumero, osoite ja ikä ovat attribuutteja.



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

Kuva 1: Esimerkki relaatiosta [Elmasri and Navathe 2004]

Yksi keskeisimmistä relaatiotietokannan ominaisuuksista on ACID (*Atomicity, Consistency, Isolation, Durability*) -sääntöjen noudattaminen. Tietokantatapahtumat eli transaktiot noudattavat näitä sääntöjä, jotka ovat atomisuus, johdonmukaisuus, erillisyys ja pysyvyys. Atomisuudella tarkoitetaan sitä, että transaktio suoritetaan yhtenä yksikkönä, jolloin se joko suoritetaan kokonaan eli kaikki siihen kuuluvat operaatiot suoritetaan tai tapahtumaa ei suoriteta lainkaan. Johdonmukaisuus tarkoittaa sitä, että transaktion tuloksena tietokannan tila siirtyy yhdestä johdonmukaisesta tilasta toiseen. Erillisyys takaa sen, että transaktio voidaan suorittaa erillisenä, muista riippumattomana eikä se aiheuta sivuvaikutuksia muissa samaan aikaan suoritettavissa tapahtumissa. Pysyvyys tarkoittaa yksinkertaisesti sitä, että transaktion tulokset tallennetaan tietokantaan pysyvästi. [Gray 1981; Haerder and Reuter 1983].

Toinen keskeisimmistä ominaisuuksista on SQL-kyselykieli. Kaikissa relaatiotietokannoissa kyselyt voidaan tehdä SQL (*Structured Query Language*) -kyselykielellä. Sen ensimmäinen versio oli nimeltään SEQUEL [Chamberlin and Boyce 1974]. Sen avulla voidaan suorittaa kaikki loogiset kyselyt sekä tietokannan luonti ja ylläpito [El-

masri ja Navathe 2004]. Vaikka SQL-kyselykieli onkin hyvin ilmaisuvoimainen, monimutkaiset oliot, XML-tiedostot ja maantieteellinen sekä ajallinen data asettavat sille haasteita [Palovska 2015].

Suosituimpia relaatiomalliin perustuvia tietokantoja ovat nykyisin Oracle [Oracle 2019], MySQL [MySQL 2019], Microsoft SQL Server [SQL Server 2019] ja PostgreSQL [PostgreSQL 2019]. MySQL on maailman toiseksi suosituin relaatiotietokanta [DB-Engines 2019]. Sillä on useita toteutuksia, kuten Standard, Enterprise ja Classic, jotka ovat perinteisiä relaatiotietokantoja. MySQL Cluster CGE -versiossa on mahdollista horisontaalinen skaalaaminen ja tietokannan taulut hajautetaan automaattisesti klusterin solmuille [MySQL 2019]. Muita ominaisuuksia ovat lineaarinen skaalautuvuus, korkea saatavuus ja transaktioiden johdonmukaisuus [MySQL 2019].

3.1 NewSQL

Termiä NewSQL käytettiin ensimmäisen kerran vuonna 2011 [Aslett 2011]. NewSQL-tietokannat ovat tietokantoja, joissa yhdistyvät relaatiotietokantojen ominaisuudet, kuten ACID-sääntöjen noudattaminen ja SQL-kielen tuki, NoSQL-tietokannoille tyypillisiin hyvään skaalautuvuuteen ja virheensietokykyyn. NewSQL-tietokantoja kutsutaan myös skaalautuviksi relaatiotietokannoiksi [Rabl *et al.* 2012]. NewSQL-tietokantoja ovat esimerkiksi Googlen Cloud Spanner [Cloud Spanner 2019], VoltDB [VoltDB 2019], MemSQL [MemSQL 2019] ja ParStream [Cisco Parstream 2018].

Cloud Spanner on Googlen kehittämä tietokanta, jossa yhdistyvät perinteisen relationaalisen tietokannan ja NoSQL-tietokannan ominaisuudet. Sillä on tyypillisiä relaatiotietokannan ominaisuuksia, kuten ennalta määrätty kaavio, SQL-kielen tuki ja vahva johdonmukaisuus. NoSQL-tietokannalle tyypillisiä ominaisuuksia taas ovat horisontaalinen skaalautuvuus, korkea saatavuus ja automaattinen replikointi. Cloud Spannerin käyttökohteita ovat esimerkiksi rahoitus- ja vakuutusalan tietokannat, joissa vaaditaan johdonmukaisuutta, sekä korkeaa saatavuutta vaativat sähköisen kaupankäynnin ratkaisut [Cloud Spanner 2019]

4 NoSQL-tietomallit

NoSQL-tietomallit ovat erilaisia relaatiotietomallista poikkeavia tietomalleja. Niitä pidetään usein uudenlaisina tietomalleina, mutta ne perustuvat vanhoihin hierarkkiseen- tai verkkotietomalliin ja niissä on piirteitä myös relaatiotietomallista. Vaikka NoSQL-tietomallit poikkeavat toisistaan monin tavoin, niille yhteisiä ominaisuuksia ovat suurten tietomäärien tehokas käsittely, hyvä horisontaalinen hajautettavuus, hyvä saatavuus sekä joustava, ennalta määräämätön kaavio tai kaaviottomuus. NoSQL-tietokantojen tavoitteena ovat myös sovelluskehityksen yksinkertaistaminen sekä sovellusten helpompi käyttöönotto [Cooper *et al*. 2010]. Avain-arvopari-, dokumentti-, sarakeperhe- ja graafitietomallin lisäksi myös oliotietomallin voidaan katsoa kuuluvan NoSQL-tietomalleihin [Nayak *et al*. 2013]. Gudivada ja muut [2014] lukevat NoSQL-tietokannoiksi lisäksi aikasarjatietokannat, XML- ja natiivi-XML- tietokannat sekä hakukoneet. Monet tietokannat ovat hybridejä eli ne toteuttavat useamman NoSQL-tietomallin piirteitä.

NoSQL-tietokannoilla ei ole yhteistä kyselykieltä, vaan tietokannat käyttävät omia, erilaisia kyselykieliään. Kyselyt voidaan toteuttaa tietokannan oman kyselykielen lisäksi ohjelmointirajapinnassa. NoSQL-tietokantojen kyselyominaisuuksien kannalta on merkittävää MapReduce-mallin tuki. MapReduce on Googlen kehittämä ohjelmointiabstraktio hyvin suurten, useille palvelimille hajautettujen tietojoukkojen käsittelyyn [Dean and Ghemawat 2008]. MapReducen avulla voidaan toteuttaa kooste- eli aggregointikyselyitä. Se mahdollistaa laskennan samanaikaisesti useilla koneilla [Hecht and Jablonski 2011]. Toiminta tapahtuu kahdessa vaiheessa. Map-vaiheessa kerätään tiedot avain-arvopareista väliarvoiksi ja Reduce-vaiheessa ne sulautetaan samalle avaimelle. Esimerkiksi sanojen määrän laskenta dokumentista voidaan toteuttaa MapReducen avulla siten, että ensimmäisessä vaiheessa lasketaan jokaisen sanan esiintymät ja tehdään niistä avain-arvopareja muodossa <sana, 1>. Toisessa vaiheessa kaikki saman sanan esiintymät lasketaan yhteen ja summa annetaan tietylle avaimelle [Dean and Ghemawat 2008]. Suurin osa dokumentti- ja sarakeperhetietokannoista ja osa avain-arvoparitietokannoista tarjoaa MapReduce-tuen.

4.1 Avain – arvoparitietomalli

Yksinkertaisin NoSQL-tietomalli on avain-arvoparitietomalli (*key-value store*). Se muistuttaa rakenteeltaan sanakirjaa tai hajautustaulua [Hecht and Jablonski 2011]. Tietomalli on havainnollistettu kuvassa 2.

Key_1	Value_1
Key_2	Value_2
Key_3	Value_1
Key_4	Value_3
Key_5	Value_2
Key_6	Value_1
Key_7	Value_4
Key_8	Value_3

Kuva 2: Avain-arvoparitietomalli [Grolinger *et al.* 2013]

Tietomalli koostuu avain-arvopareista. Avaimet ovat uniikkeja merkkijonoja, joiden perusteella avainta vastaava arvo tunnistetaan. Arvo voi olla esimerkiksi kokonaisluku, merkkijono, taulukko tai objekti [Grolinger *et al.* 2013]. Tietomalli on täysin kaavioton ja ainoa tapa muodostaa jonkinlainen rakenne on avain-arvoparien ryhmittely kokoelmiksi [Hecht and Jablonski 2011].

Avain-arvoparitietokantojen ohjelmointirajapinnat mahdollistavat vain yksinkertaiset luonti- (*put*), luku- (*get*) ja poisto-operaatiot (*delete*). Monimutkaisemmat kyselyt voidaan toteuttaa vain sovellustasolla. Avain-arvoparitietomallissa kyselyjä voi suorittaa vain avaimen perusteella, koska arvot ovat tietokannalle täysin läpinäkyviä. Lisäksi vain avaimet voidaan indeksoida. [Hecht and Jablonski 2011]

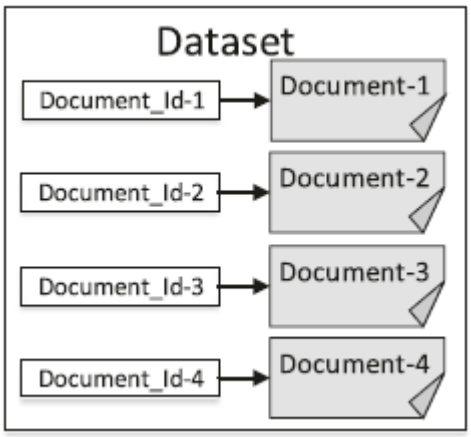
Avain-arvoparitietokannat soveltuvat yksinkertaisten operaatioiden suorittamiseen. Sen sijaan ne eivät sovi käyttötapauksiin, joissa tarvitaan monimutkaisia kyselyitä, useiden tietueiden samanaikaista hakua, arvon perusteella tapahtuvaa hakua tai tietojen välisten suhteiden ja rakenteiden käsittelyä. Avain-arvoparitietokannat skaalautuvat hyvin klusterin solmujen kesken [Gudivada *et al.* 2014] ja ovat tehokkaita tiedon tallentamisessa hajautettuun tietokantaan. Avain-arvoparitietokantoja käytetään usein varsinaisen tietovaraston välimuistina [Grolinger *et al.* 2013]. Ne sopivat myös suurten tietomäärien nopeaan reaaliaikaiseen käsittelyyn. Käyttökohteita ovat esimerkiksi sanakirjat, viestipalvelut, sähköisen kaupankäynnin suosittelujärjestelmät ja ostoskorit sekä käyttäjäistuntojen tietojen tallennus. Avain-arvoparitietokannat voidaan jakaa kahteen ryhmään sen mukaan, käsitelläänkö tieto muistissa (*in-memory*) vai levyllä (*persistent*).

Avain-arvoparitietokantoja ovat esimerkiksi Redis [Redis 2019], Riak KV [Riak 2019], Voldemort [Voldemort 2019], Scalaris [Scalaris 2018], Oracle NoSQL [Oracle NoSQL 2019] ja Tarantool [Tarantool 2019]. Suosituin avain-arvoparitietokanta on Redis [DB-Engines 2019]. Se on avoimeen lähdekoodiin perustuva, muistissa toimiva tietorakennevarasto. Tuetut tietorakenteet ovat merkkijonot, hajautustaulut, listat, joukot, lajitellut joukot. Tietokannan käsittelyyn on olemassa useita komentoja. Peruskomennot ovat *set*, *get* ja *del* eli arvon asettaminen avaimelle, arvon hakeminen sekä avaimen poistaminen. Redistä voidaan käyttää sekä tietokantana, välimuistina että viestien välityksessä.

Koska se toimii kokonaan muistissa, on operaatioiden suorittaminen nopeaa. Toisaalta tämä rajoittaa käsiteltävän tietojoukon kokoa, koska se on riippuvainen käytettävissä olevasta muistitilasta. Redistä käytetään usein yhdessä relaatiotietokannan tai muun levyllä toimivan tietokannan kanssa, jolloin se toimii välimuistina tietojoukon osan käsittelyssä. [Redis 2019]

4.2 Dokumenttivarasto

Dokumenttivaraston (*Document Store*) rakenne koostuu dokumenteista, jotka kuvaavat yhtä entiteettiä ja sen ominaisuuksia. Dokumenttivaraston tietomalli on havainnollistettu kuvassa 3. Dokumentit tunnistetaan tietovaraston sisällä yksilöllisen avaimen eli tunnisteen (id) perusteella [Hecht and Jablonski 2011]. Joissain dokumenttitietokannoissa, kuten MongoDB:ssa, dokumentit voidaan ryhmitellä kokoelmiin, jolloin avaimen tulee olla uniikki kokoelman sisällä [MongoDB 2019]. Dokumentin sisältö muodostuu avain-arvopareja sisältävistä kentistä. Avain on tyypiltään merkkijono ja kentän arvo voi olla esimerkiksi merkkijono, numeerinen arvo tai lista. Kentät voivat sisältää myös sisäkkäisiä objekteja, jolloin dokumentin sisältö muodostaa hierarkkisen, puumaisen rakenteen [Gudivada *et al*. 2014]. Dokumenttitietomalli muistuttaakin vanhaa hierarkkista tietomallia.



Kuva 3: Dokumenttivaraston tietomalli [Grolinger *et al.* 2013]

Dokumenttivarasto mahdollistaa avain-arvoparitietomallia monimutkaisemmat tietorakenteet. Dokumenttien välisiä suhteita voi kuitenkin hallita vain sovelluskerroksessa, poikkeuksena sisäkkäiset dokumentit [Hecht and Jablonski 2011]. Dokumenttivaraston tietomalli on yleensä kaavioton, jolloin dokumentit voivat olla rakenteeltaan erilaisia [Grolinger *et al.* 2013]. Myöskään kenttien arvoalueita ei ole ennalta määritelty. Dokumentit voivat kuitenkin olla myös rakenteisia tai puolirakenteisia [Elmasri and Navathe 2004]. Rakenteiset dokumentit ovat määrämuotoisia, kun taas puolirakenteisten doku-

menttien rakennetta ei ole ennalta määritelty, vaan ne voivat poiketa toisistaan. Dokumentit esitetään yleensä käyttäen JSON (*JavaScript Object Notation*) -notaatiota, XML:ää (*Extensible Markup Language*) tai JSON:ista johdettua formaattia kuten BSON:ia (*Binary JSON*). JSON tukee erilaisia tietotyyppejä, kuten numerot, taulukot ja boolean-arvot [Hashem and Ranc 2016].

Toisin kuin avain-arvoparitietomallissa, tietoa voidaan hakea avainten lisäksi myös arvojen perusteella, sillä ne eivät ole järjestelmälle läpinäkyviä [Hecht and Jablonski 2011]. Sekä avaimet että arvot on mahdollista indeksoida [Grolinger *et al.* 2013]. Näin ollen on mahdollista suorittaa avain-arvoparitietomallia monipuolisempia kyselyitä. Yksinkertaisten lisäys-, luku- ja poisto-operaatioiden lisäksi kyselyitä voidaan kohdistaa myös tietylle arvovälille sekä sisäkkäisille dokumenteille [Hecht and Jablonski 2011]. Dokumenttitietokannan ohjelmointirajapinta mahdollistaa dokumenttien ja kenttien ensi- ja toissijaisen indeksoinnin. Tietokannat tarjoavat omia, myös SQL-tyyppisiä kyselykieliä. XML-tietokannoissa voidaan käyttää kyselykielenä esimerkiksi XPathia (*XML Path Language*) [Clark and DeRose 1999] tai XQueryä [Chamberlin 2002].

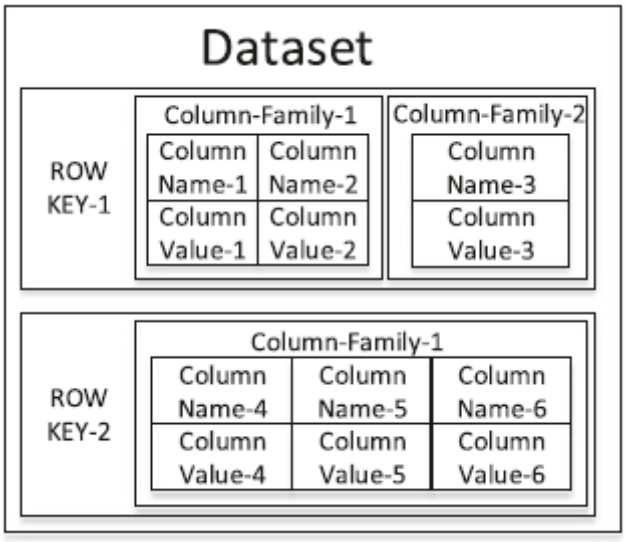
Dokumenttitietokanta sopii sovelluksiin, joissa tieto voidaan luontevasti esittää dokumenttimuodossa ja jossa on tarvetta avain-arvoparitietomallia monimutkaisemmille rakenteille ja kyselyille. Käyttökohteita ovat esimerkiksi joustavat web-sivut kuten blogit, reaaliaikaisen analytiikan järjestelmät, kirjautumisen toteutukset ja vapaan tekstin hakeminen.

Dokumenttitietokantoja ovat esimerkiksi MongoDB [MongoDB 2019], RavenDB [RavenDB 2019], CouchDB [CouchDB 2019] ja Couchbase [Couchbase 2019]. XML-formaattia käyttää esimerkiksi Oracle Berkeley DB [Oracle Berkeley DB 2019]. Myös hakukonetietokanta Elasticsearch [Elasticsearch 2019] sekä dokumentti- ja graafitietokannan hybridi OrientDB [OrientDB 2019] voidaan luokitella dokumenttitietokannoiksi.

Suosituin dokumenttitietokanta on MongoDB [DB-Engines 2019]. Se on hajautettu tietokanta, jossa dokumentit voidaan tallentaa yhteen tai useampaan kokoelmaan [MongoDB 2019]. Tieto tallennetaan BSON-dokumentteihin. MongoDB:lla on oma, SQL-tyyppinen kyselykielensä. Muita ominaisuuksia ovat ad hoc- kyselyt, indeksointi ja reaaliaikainen aggregointi [MongoDB 2019].

4.3 Sarakeperhe

Sarakeperheen (*Column Family, Wide-Column Store*) tietomalli on havainnollistettu kuvassa 4. Tietomalli koostuu riveistä, sarakkeista ja sarakeperheistä. Rivit tunnistetaan yksilöllisen avaimen perusteella. Rivi sisältää sarakeperheitä ja sarakeperhe sisältää sarakkeita. Sarakeperhettä voidaan käsitellä yhtenä yksikkönä. Sarake sisältää avain-arvopareja. Arvojen kaikki versiot tallennetaan kaikki aikajärjestyksessä [Hecht and Jablonski 2011].



Kuva 4: Sarakeperheen tietomalli [Grolinger *et al.* 2013]

Taulukkomainen rakenne muistuttaa hieman relaatiotietomallia. Rivien ei kuitenkaan tarvitse olla rakenteeltaan samanlaisia, vaan ne voivat sisältää eri määrän erilaisia sarakeperheitä ja sarakkeita [Grolinger *et al.* 2013]. Tietorakenne ei ole niin joustava kuin avain-arvopari- tai dokumenttitietomallissa, sillä sarakeperheet määritellään usein tietokannan käynnistämisen yhteydessä [Hecht and Jablonski 2011]. Uusia sarakkeita ja rivejä voidaan lisätä suoritusaikana [Hecht and Jablonski 2011].

Sarakeperhetietomallissa indeksointi ja kyselyt voidaan kohdistaa avaimen lisäksi sarakeperheisiin ja sarakkeisiin, mikä mahdollistaa avain-arvoparitietomallia monipuolisemmat kyselyt. Perusoperaatioiden lisäksi voidaan suorittaa esimerkiksi tiettyyn arvoväliin kohdistuvia kyselyitä. Tiedonhaku on nopeaa erityisesti sarakeperheen sisällä, koska siihen kuuluvat sarakkeet tallennetaan samaan palvelinsolmuun eli fyysisesti hyvin lähelle toisiaan. Myös palvelimen vertikaalinen skaalaaminen voi siksi olla hyödyllistä.

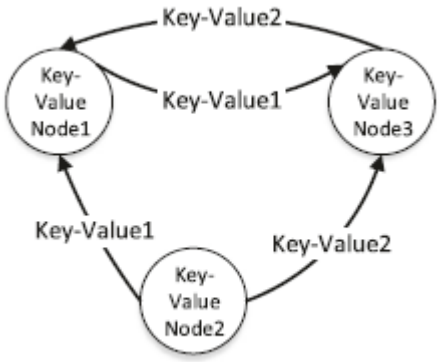
Sarakeperhetietokannoilla on omia, SQL-tyyppisiä kyselykieliä kuten Cassandran CQL [Cassandra 2016] ja Hypertablen HQL [Hypertable 2014]. Tietojen välisiä suhteita ei ole mahdollista käsitellä, vaan ne on toteutettava sovellustasolla [Grolinger *et al.* 2013]. Lähes kaikki sarakeperhetietokannat tukevat MapReducea.

Sarakeperhetietomalli sopii erittäin suurten, hajautettujen tietokantojen käsittelyyn ja erityisesti sellaiselle datalle, jossa attribuuttien määrä vaihtelee riveittäin [Hecht and Jablonski 2011]. BigTablea käytetään esimerkiksi web-indeksoinnissa ja -analytiikassa, personoitujen hakujen toteuttamisessa, sähköpostisovellus Gmailissa ja Google Earthissa [Chang *et al.* 2008]. Muita sarakekeperhetietokantoja ovat esimerkiksi DynamoDB [DeCandia *et al.* 2007], Cassandra [Cassandra 2016], HBase [HBase 2019] ja Hypertable [Hypertable 2014].

Suosituin sarakeperheistä on Apache Cassandra [DB-Engines 2019]. Cassandra on Facebookin kehittämä toisen sukupolven sarakeperhetietokanta, joka on suunniteltu erittäin suurten tietomäärien käsittelyyn hajautetussa palvelinympäristössä. Se sisältää lisäulottuvuuden nimeltään supersarake, jonka alle sarakkeita voidaan ryhmitellä. Tämä ominaisuus mahdollistaa muita sarakeperheitä monimutkaisempien ja ilmaisuvoimaisempien tietorakenteiden käsittelyn. [Cassandra 2016]

4.4 Graafitietomalli

Graafitietomalli muistuttaa rakenteeltaan verkkotietomallia. Erona muihin NoSQL-tietomalleihin, graafitietokannassa olennaisinta on tiedon välisten suhteiden hallinta. Graafitietomalli on havainnollistettu kuvassa 5.



Kuva 5: Esimerkki graafitietomallista [Grolinger et al. 2013]

Tietomalli koostuu graafeista, jotka sisältävät joukon entiteettejä kuvaavia solmuja ja niitä yhdistäviä suunnattuja kaaria [Grolinger *et al.* 2013]. Solmuihin saapuu ja niistä lähtee joukko kaaria. Solmuilla ja kaarilla on yksilölliset tunnisteet. Ne sisältävät tietoa avain-arvoparimuodossa. Kaaren tunniste kertoo alku- ja loppusolmun välisen suhteen tyypin. Tietokantaoperaatioissa hyödynnetään graafeille tyypillistä kulkemista (*traversal*), jossa käydään läpi solmut ja niiden suhteet annettujen sääntöjen mukaisesti [Angles and Gutierrez 2008].

Graafitietokannat sopivat käyttökohteisiin, joissa on tarvetta kuvata verkkomaisia tietorakenteita ja tietojen välisiä riippuvuuksia. Tällaisia ovat esimerkiksi sijaintitiedon hallinta sekä navigointi- ja suosittelujärjestelmät. Graafitietokantoja ovat esimerkiksi Neo4j [Neo4j 2019] ja GraphDB [GraphDB 2019]. Graafitietokannaksi voidaan luokitella myös dokumentti- ja graafitietokannan hybridi OrientDB [OrientDB 2019].

Suosituin graafitietokanta on Neo4j [DB-Engines 2019]. Neo4j on natiivi graafitietokanta voimakkaasti toisiinsa liittyvän datan hallintaan. Sen avulla voidaan analysoida tietoa ja tietojen välisiä yhteyksiä piilotettujen suhteiden löytämiseksi. Neo4j on täysin ACID-yhteensopiva. Kyselykielinä voidaan käyttää Cypheriä, Gremliniä tai SparkQL:ia. [Neo4j 2019]

5 Tietokantojen hajautuksesta

Tietokantoja ylläpidetään palvelimilla. Tietokannan koon kasvaessa täytyy joko kasvattaa tietokantapalvelimen kapasiteettia tai lisätä palvelimia. Palvelimen kapasiteetin kasvattamista ja sen ominaisuuksien parantamista kutsutaan vertikaaliseksi skaalaamiseksi [Bazar and Iosif 2014]. Horisontaalisella skaalaamisella taas tarkoitetaan uusien, usein halpojen palvelinlaitteiden lisäämistä palvelinten muodostamaan verkkoon eli klusteriin [Hashem and Ranc 2016]. Klusterin palvelimia kutsutaan myös solmuiksi. Tietokantapalvelimet toteutetaan nykyään yhä useammin pilvipalveluna.

Relaatiotietokannoissa skaalautuvuus on pääasiassa vertikaalista. Myös horisontaalinen skaalaaminen on mahdollista, mutta esimerkiksi tiedon johdonmukaisuuden säilyttäminen hajautetuissa järjestelmissä on vaikeaa [Abramova *et al.* 2014b], eikä palvelinten määrän kaksinkertaistaminen kaksinkertaista järjestelmän suorituskykyä [Hecht and Jablonski 2011]. Lisäksi taulujen väliset liitosoperaatiot ja ACID-ominaisuuksista johtuvat lukitusmekanismit heikentävät relaatiotietokantojen suorituskykyä hajautetussa järjestelmässä [Hecht and Jablonski 2011]. NoSQL-tietokannat sen sijaan ovat suunniteltu toimimaan hajautetussa palvelinarkkitehtuurissa ja horisontaalinen hajautus on niissä automaattista. Tietokantojen hajauttamiseen on erilaisia tapoja ja mekanismeja. Tässä luvussa käsitellään hajautustapojen ja -mekanismien lisäksi niiden vaikutusta saatavuuteen, vikasietoisuuteen ja johdonmukaisuuteen.

5.1 CAP-teoreema ja BASE-oikeellisuusmalli

Hajautetuissa järjestelmissä voidaan CAP (*Consistency, Availability, Partition tolerance*) -teoreeman mukaan toteuttaa vain kaksi kolmesta ominaisuudesta, jotka ovat johdonmukaisuus, saatavuus ja osioinninsietokyky. Johdonmukaisuudella tarkoitetaan sitä, että tiedosta ei ole samanaikaisesti olemassa useita versioita, vaan kaikki solmut sisältävät saman version. Korkea saatavuus takaa sen, että tieto on aina saavutettavissa. Osioinninsietokyky tarkoittaa sitä, järjestelmä sietää yhteyden katkeamisen solmujen väliltä. Teoreeman mukaan hajautetussa järjestelmässä voidaan siis valita joko johdonmukaisuus tai saatavuus. CAP-teoreeman tarkoituksena oli, että nähtäisiin erilaisia tapoja toteuttaa järjestelmiä. Sääntö kuitenkin yksinkertaistaa liikaa ominaisuuksien välisiä riippuvuuksia, joilla on enemmän merkitystä nykypäivänä kuin ennen. Käsittelemällä järjestelmän hajautettuja osioita eksplisiittisesti, on mahdollista saavuttaa piirteitä kaikista kolmesta ominaisuudesta optimoimalla niiden johdonmukaisuutta ja saavutettavuutta. CAP-teoreeman vastaisia ovat ainoastaan sellaiset hajautetut järjestelmät, joissa on täydellinen saatavuus ja johdonmukaisuus. [Brewer 2012]

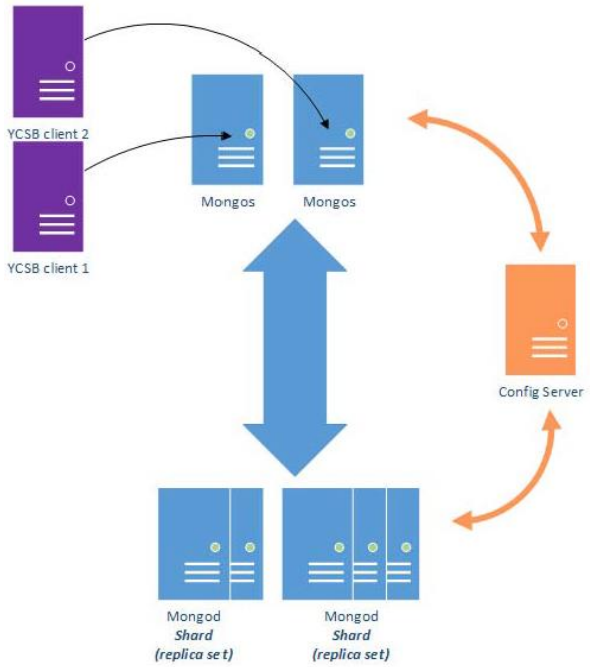
NoSQL-tietokannoissa noudatetaan ACID-sääntöjen sijaan BASE-oikeellisuusmallia. BASE (*Basically Available, Soft state, Eventually consistent*) -ominaisuudet ovat: aina saatavilla, ei-aina-oikeellinen ja lopulta johdonmukainen [Brewer 2012]. ACID ja

BASE ovat vastakkaiset johdonmukaisuuden ja saatavuuden suhteen. ACID-järjestelmät ovat johdonmukaisia, koska niissä kaikki isäntäpalvelimen kopiot päivitetään samaan aikaan eli synkronisesti. Tämä tarkoittaa kuitenkin myös sitä, että järjestelmä ei ole saatavilla ennen kuin kaikki päivitykset on suoritettu loppuun. BASE-järjestelmissä kopiot sen sijaan päivitetään asynkronisesti eli vuorotellen tietyin väliajoin, jolloin kopiot saattavat olla hetkittäin epäjohdonmukaisia, mutta järjestelmä on aina saatavilla [Hecht and Jablonski 2011]. NoSQL-tietokannat eivät kuitenkaan ole välttämättä puhtaasti BASE-järjestelmiä, vaan niissä voi olla myös ACID-ominaisuuksia.

5.2 Replikointi ja johdonmukaisuus

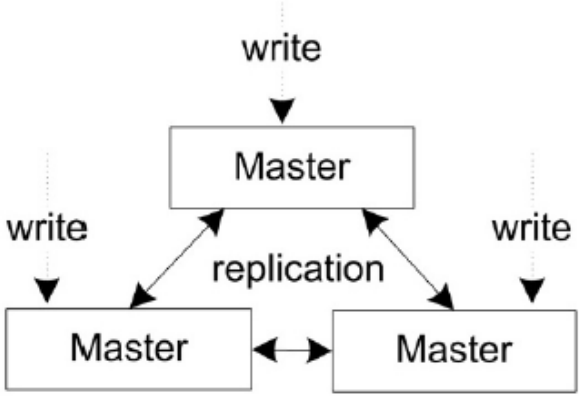
Replikointi eli toisintaminen tarkoittaa tietojen kopioimista useammille solmuille [Hecht and Jablonski 2011]. Replikointi lisää luku- ja kirjoitusoperaatioiden skaalautuvuutta sekä järjestelmän luotettavuutta [Grolinger *et al.* 2013]. Replikointi parantaa myös vikasietoisuutta ja saatavuutta, koska solmun vikaantuessa liikenne voidaan ohjata sen kopiolle ja palauttaa siltä mahdollisesti kadonnut tieto [Hecht and Jablonski 2011]. Replikointi voi tapahtua joko synkronisesti tai asynkronisesti. Synkronisen replikoinnin hyötynä on, että kaikki solmut ovat aina ajan tasalla. Toisaalta se voi hidastaa päivitysoperaatioita. Samoin saatavuus voi kärsiä, jos päivitys ei onnistu osan solmuista ollessa tilapäisesti pois käytöstä. Asynkroninen replikointi taas sallii replikoiden tiedon vanhentumisen. Lisäksi virhetilanne voi aiheuttaa tiedon menetystä, jos päivitys keskeytyy ennen kuin replikointi on suoritettu [Cooper *et al.* 2010].

NoSQL-tietokannoissa käytetään pääsiassa kolmea erilaista replikointiratkaisua: isäntä-orja, isäntä-isäntä ja isännätön. Näissä tietokanta hajautetaan eri tavoin palvelinsolmujen kesken. Isäntä-orja -arkkitehtuuria käyttävät esimerkiksi MongoDB, Redis, BerkeleyDB ja HBase. Esimerkkinä isäntä-orja -arkkitehtuurista on esitetty MongoDB:n arkkitehtuuri kuvassa 6.



Kuva 6: MongoDB:n isäntä-orja -arkkitehtuuri [Gandini *et al.* 2014]

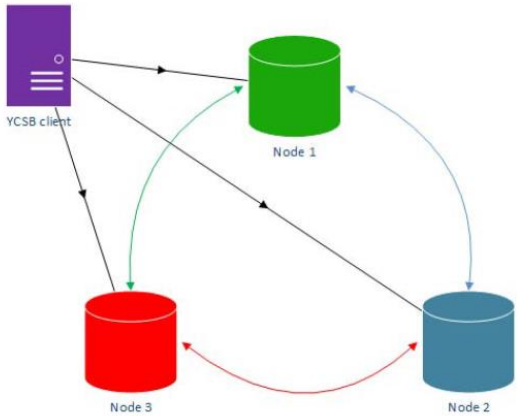
Tällaisessa arkkitehtuurissa yksi solmuista on isäntä ja muut orjia. Isäntä voi suorittaa sekä luku- että kirjoitusoperaatioita, mutta orjat pelkkiä lukuoperaatioita. Isäntä replikoidaan orjasolmuille [Grolinger *et al.* 2013]. MongoDB:n arkkitehtuurissa on kolmentyyppisiä solmuja: mongod-solmut, mongos-solmut ja konfiguraatiopalvelimet. Mongod-solmujen tehtävänä on tiedon varastointi ja haku, mongos-solmut kommunikoivat klusterin ulkopuolelle ja konfiguraatiopalvelimet sisältävät tallennettujen tietojen metadatan, jota tarvitaan silloin, jos solmu vikaantuu [MongoDB 2019]. Isäntä-orja -arkkitehtuurin heikkous on sen virhealttius, sillä koko klusterin saatavuus on riippuvainen reitittävästä solmusta [Gandini *et al.* 2014]. Virhealttiuden vähentämiseksi reititinsolmu voidaan replikoida useammille palvelimille.



Kuva 7: Isäntä-isäntä -replikointi [Grolinger *et al.* 2013]

Isäntä-isäntä -arkkitehtuurissa kirjoituspyyntöjä voi käsitellä useampi solmu. Esimerkki arkkitehtuurista on esitetty kuvassa 7. Kun pyyntö on käsitelty joissakin solmuista, kyselyt levitetään muille solmuille. Isäntä-orja -arkkitehtuurista poiketen pyynnöt voivat siis kulkea solmujen välillä eri tavoin. Isäntä-isäntä -arkkitehtuuria käyttävät esimerkiksi CouchDB ja Couchbase. [Grolinger *et al.* 2013]

Isännätöntä arkkitehtuuria kutsutaan myös rengastopologiaksi [Tang and Fan 2016]. Siinä palvelinklusterin solmut muodostavat vertaisverkon, jossa solmut ovat identtisiä ja niillä kaikilla on samanlainen rooli [Tang and Fan 2016]. Tällainen arkkitehtuuri on käytössä esimerkiksi Cassandrassa, Voldemortissa ja Riakissa. Rengastopologia on esitetty kuvassa 8.



Kuva 8: Rengastopologia [Gandini *et al.* 2014]

Tällaisen arkkitehtuurin etuna on se, ettei yhden solmun vikaantuminen haittaa järjestelmän toimintaa. Solmut myös tietävät toistensa tilan sekä tiedon sijainnin [Tang and Fan 2016]. Tietokannan ulkopuolelle kommunikoivat solmut voidaan määritellä ja mikä tahansa solmuista voi ottaa reitittäjän rooli [Tang and Fan 2016]. Jokaiselle tietokantaan lisättävälle tietueelle lasketaan hajautusarvo, ja tietty hajautusarvojen arvoväli jaetaan

solmujen kesken tasaisesti. Solmut voidaan replikoida toisiin solmuihin ja replikoiden määrä voidaan asettaa replikointikertoimen avulla [Gandini *et al*. 2014]. Replikointi voidaan toteuttaa joko synkronisesti tai asynkronisesti ja replikointitapa voidaan valita jokaiselle päivitysoperaatiolle erikseen. Sekä kyselyiden suoritus että tiedon hallinta suoritetaan kaikissa solmuissa, mikä parantaa tietokannan tehokkuutta [Tang and Fan 2016].

NoSQL-tietokantojen johdonmukaisuuden taso vaihtelee. Tietokanta voi olla täysin johdonmukainen, lopulta johdonmukainen tai jotain näiden väliltä. Taso voi olla myös konfiguroitavissa. Tällaisen niin sanotun optimistisen replikaation avulla voidaan määritellä niiden solmujen määrä, joiden täytyy suorittaa loppuun kirjoitusprosessit tai vastata lukupyyntöön. Optimistisen replikoinnin tarjoavat esimerkiksi Voldemort, Riak, MongoDB ja Cassandra. Johdonmukaisuus vaikuttaa järjestelmän saatavuuteen. Lopulta johdonmukaisissa järjestelmissä on parempi saatavuus kuin täyden johdonmukaisuuden tarjoavissa. Lopulta johdonmukaisia tietokantoja ovat esimerkiksi Redis, CouchDB ja Neo4j. Täyden johdonmukaisuuden tarjoavissa tietokannoissa kuten HBase ja Hypertable, ei käytetä replikointia kuorman tasapainottamisessa [Hecht and Jablonski 2011].

5.3 Sirpalointi

Sirpaloinnilla (*sharding*) tarkoitetaan tiedon hajauttamista palvelinklusterin solmuille. Jokainen sirpale on yhden tai useamman solmun ryhmä, jota kutsutaan kopiojoukoksi eli replikaksi [Gandini *et al.* 2014]. Sirpaleessa kaikki solmut sisältävät samat tiedot. Solmujen määrä sirpaleessa on järjestelmän replikointikerroin. Sirpalointi voidaan tehdä joko tietueiden avainten arvovälin perusteella tai johdonmukaisella hajautuksella (*consistent hashing*). Arvoväliin perustuvassa osituksessa yksi reitittävä solmu vastaa osituksesta. Se jakaa tietuevälit lohkoiksi tietueiden avainten perusteella ja välittää ne muille klusterin solmuille varastoitaviksi. Arvoväliin perustuvaa hajautusta käytetään esimerkiksi MongoDB:ssa, BigTablessa, HBasessa ja Hypertablessa. Luonnollisesti tiettyyn arvoväliin kohdistuvat kyselyt toimivat tällä tavoin hajautetussa järjestelmässä nopeasti. Johdonmukaisessa hajautuksessa tietueet hajautetaan solmuille hajautusfunktion avulla. Reitittävää solmua ei ole, vaan jokainen solmu vastaa tietystä hajautusalueesta. Tietueiden sijainti klusterissa on helppo laskea niiden avainten sijainnin perusteella, mikä tekee hausta nopeaa. Sen sijaan avainten hakeminen tietyltä arvoväliltä voi olla hidasta, sillä hajautus tapahtuu sattumanvaraisesti. Johdonmukaista hajautusta käytetään esimerkiksi Riakissa, CouchDB:ssa sekä Cassandrassa. [Hecht and Jablonski 2011].

6 Tietokantojen suorituskyky

Tietokannan suorituskyvyn mittareina käytetään yleensä viivettä (*latency*), suoritusaikaa (*execution time*) ja suoritustehoa (*throughput*). Viiveellä tarkoitetaan aikaa, joka kuluu pyynnön lähettämisestä sovelluksesta vastauksen saamiseen tietokannalta [Klein *et al.* 2015]. Suoritusajassa ei huomioida yhteyden muodostamiseen kuluvaa aikaa. Suoritusteholla tarkoitetaan operaatiomäärää, jonka tietokanta pystyy suorittamaan tietyssä ajassa tai ennen tietokannan saturaatiota. Suoritusteho lasketaan jakamalla suoritettujen operaatioiden kokonaismäärä suoritusajalla [Klein *et al.* 2015]. Viive ja suoritusteho ovat vastakkaiset mittarit. Silloin, kun järjestelmän suorittamien operaatioiden määrä kasvaa, kasvaa myös yksittäisten operaatioiden viive. Tämä johtuu muun muassa levyn, prosessorin ja verkon käytön lisääntymisestä [Cooper *et al.* 2010]. Mitä parempi järjestelmän suorituskyky, sitä vähemmän tarvitaan palvelimia riittävän suoritustehon saavuttamiseen hyväksyttävän viiveen puitteissa [Cooper *et al*. 2010]. Suorituskyvyn lisäksi voidaan tarkastella myös skaalautuvuutta, elastisuutta ja saatavuutta.

6.1 Suorituskyvyn mittaaminen

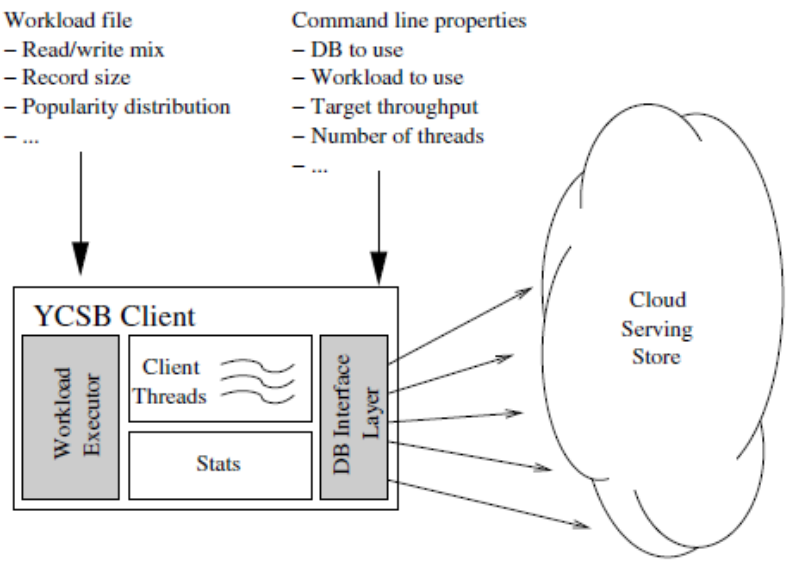
Tietokantojen suorituskyvyn vertailu voi olla hankalaa. Suorituskyvyltään parhaan mahdollisen tietokannan valinta voi vaatia useamman tietokannan suorituskyvyn mittausta ja mittaustulosten vertailua. Useiden tietokantojen lataaminen ja manuaalinen testaaminen on kuitenkin hidasta ja kallista. Lisäksi tuotantokäyttöä vastaavan testiympäristön luomiseksi saatettaisiin tarvita prototyyppi, joka sisältää valtavan määrän sekä tietoa että palvelimia. Suorituskyvyn mittaamiseen ja vertailuun on kuitenkin olemassa useita työkaluja. Avain-arvoparidataa sisältävien tietokantojen testaamiseen sopivia työkaluja ovat esimerkiksi Yahoo! Cloud Serving Benchmark [Cooper *et al.* 2010] ja AquaFold Aqua Studio [Aqua Studio 2019]. Graafitietokantojen suorituskyvyn mittaamiseen voidaan käyttää esimerkiksi XGDBench:iä [Dayarathna and Suzumura 2014].

Saatavuutta voidaan testata keskeyttämällä palvelimen toiminta kesken suorituksen ja tutkimalla sitten, onko keskeyttäminen aiheuttanut virheitä ja mitkä ovat sen vaikutukset tiedon oikeellisuuteen ja saatavuuteen. Skaalautuvuutta ja elastisuutta voidaan tutkia lisäämällä palvelimia ja tutkimalla toimenpiteen vaikutusta järjestelmän suorituskykyyn. Järjestelmän skaalautuvuus on hyvä, jos suorituskyky pysyy suhteellisesti vakiona, kun palvelinten määrää, datamäärää tai kuormitusta kasvatetaan. Hyvän skaalautuvuuden saavuttamiseksi on tärkeää tasapainottaa kuormaa palvelinten välillä sekä välttää pullonkauloja [Cooper *et al.* 2010]. Lisäksi järjestelmän kapasiteettia tulisi voida kasvattaa ilman sovellusmuutoksia. Toisaalta resurssien määrän tulisi olla myös vähennettävissä, jos järjestelmän kuormitus vähenee.

6.2 Yahoo Cloud Serving Benchmark

Suosituin työkalu NoSQL-tietokantojen suorituskyvyn mittaamiseen ja vertailuun on YSCB (*Yahoo Cloud Serving Benchmark*) [Cooper *et al.* 2010]. YCSB on avoimeen lähdekoodiin perustuva sovelluskehys ja vertailukehittämisen työkalu. Suorituskyvyn lisäksi YCSB:n avulla voidaan tutkia myös skaalautuvuutta, elastisuutta, saatavuutta ja replikointia. Elastisuudella tarkoitetaan sitä, että tietokannan suorituskyky kasvaa lineaarisesti uusia koneita lisättäessä ja tietokantamuutokset voidaan tehdä ilman sovelluksen keskeytystä [Cooper *et al.* 2010].

YCSB:n rakenne on esitetty kuvassa 9. Asiakas (*YCSB Client*) on Java-kielellä kirjoitettu ohjelma, jonka avulla generoidaan sekä tietueet että operaatiot. Se koostuu kuormitusgeneraattorista, asiakassäikeistä, tietokantarajapinnasta sekä tilastoista, joihin mittaustulokset tallennetaan. Asiakassäikeiden avulla simuloidaan tietokannan samanaikaisia käyttäjiä.



Kuva 9: YCSB:n rakenne [Cooper *et al.* 2010]

Säikeiltä tulevat pyynnöt muutetaan tietokantakutsuiksi tietokantarajapinnassa. Rajapinta riippuu käytettävästä tietokannasta. Yksi yleisimmistä rajapinnoista on REST [Fielding 2000]. Asiakkaalle kerrotaan komentojonoparametrein esimerkiksi käytettävä tietokanta, kuormitus, säikeiden määrä ja suoritustehon tavoitetaso.

Asiakas käsittelee kuormatiedoston, joka sisältää erilaisia vakiokuormituksia (*YCSB Core Package*). Kuormatiedostot sisältävät tietyn jakauman mukaisesti tietokantaoperaatioita. Kuormitusgeneraattori määrittelee tietojoukon, lataa sen tietokannan solmuille sekä huolehtii operaatioiden suorittamisesta. Operaatiot voivat olla lisäys-, päivitys-, luku- tai skannausoperaatioita ja ne kohdistetaan kohteille eli tietokannan tietuille

satunnaisesti. Operaation ja tietueen valinta tehdään kullekin kuormitukselle määritellyn jakauman mukaisesti. Käytettävät jakaumat ovat yhdenmukainen, Zipfin jakauma, viimeisimmät ja monimuotoinen. Yhdenmukaista jakaumaa käytettäessä kaikki kohteet ovat samanarvoisia. Zipfin jakaumassa jotkin kohteet saavat enemmän painoarvoa kuin toiset. Viimeisimmät-jakaumassa annetaan suurempi painoarvo viimeksi lisätyille tietueille. Monimuotoisessa jakaumassa voidaan asettaa kohteille todennäköisyys, joiden perusteella ne tulevat valituksia. [Cooper *et al.* 2010]

Vakiokuormitukset ovat eri kokoisia avain-arvoparitietueita sisältäviä tauluja. Tietueet koostuvat pääavaimesta sekä ennalta määriteltävästä määrästä kenttiä. Pääavain on tietotyypiltään merkkijono (esimerkiksi ”*user123456”).* Kentät on nimetty juoksevasti (esimerkiksi ”*field0*”, *”field1”..., ”fieldn”*) ja niiden arvot ovat satunnaisia ASCII-merkkijonoja. Vakiokuormitusten lisäksi kuormituksia on mahdollista konfiguroida joko parametreja tai ohjelmakoodia muuttamalla. Vakiokuormitukset ja niissä käytettävät jakaumat on esitetty taulukossa 1. [Cooper *et al.* 2010]

Tunnus	**Kuvaus**	**Operaatiot**	**Tietueen valintaperuste**
A	Päivityspainotteinen	50 % luku, 50 % päivitys	Zipfin jakauma
B	Lukupainotteinen	95 % luku, 5 % päivitys	Zipfin jakauma
C	Pelkkä luku	100 % luku	Zipfin jakauma
D	Viimeisimpien luku	95 % luku, 5 % lisäys	Viimeksi lisättyjen painotus
E	Lyhyet välit	95 % skannaus, 5 % lisäys	Ensimmäinen tietue: Zipfin jakauma; skannattavat tietueet: yhdenmukainen jakauma
F	Luku- / luku-muokkaus-kirjoitus	50 % luku, 50 % luku-muokkaus-kirjoitus	-
G	Päivityspainotteinen	5 % luku, 95 % päivitys	-
H	Pelkkä kirjoitus	100 % kirjoitus- / päivitysoperaatioita	-

Taulukko 1: YCSB:n vakiokuormitukset [Cooper *et al.* 2010]

7 Tietokantojen suorituskyvyn vertailu

NoSQL-tietokannat on suunniteltu suurten tietomäärien käsittelyyn hajautetuissa järjestelmissä. Niiden suorituskykyä voidaan kuitenkin vertailla myös yksinkertaisemmilla arkkitehtuureilla ja pienemmillä tietuemäärillä, sillä vaikka koejärjestelyt eivät vastaisikaan todellisten sovellusten tarpeita, voidaan niiden avulla saada suuntaa-antavaa tietoa tietokantojen suorituskyvystä erityyppisillä kuormituksilla ja toisiinsa verrattuna. Kohdassa 7.1 käsitellään tällaisia tutkimuksia. Lähes kaikissa käytettiin YCSB:ia ja sen standarditietueita ja -kuormituksia.

Kohdassa 7.2 käydään läpi tapaustutkimuksia, joissa suorituskykyä tutkitaan tiettyä, määriteltyä käyttökohdetta varten. Tällöin tietokannalle on asetettu yksityiskohtaisempia vaatimuksia. Vaikka käyttötapaustutkimukset on tehty spesifiä käyttökohdetta varten, saadaan niistä myös yleistettävää tietoa tietokantojen suorituskyvystä. Esimerkiksi käytettävä data ja tietomäärä sekä samanaikaisten käyttäjien määrä vastaavat usein paremmin todellisia sovelluksia kuin kohdan 7.1. kokeelliset tutkimukset.

7.1 Suorituskyky kokeellisissa vertailututkimuksissa

Abramovan ja Bernardinon [2013] tutkimuksessa mitattiin dokumenttitietokanta MongoDB:n ja sarakeperhe Cassandran suorituskykyä YCSB:n avulla. Kuormituksina käytettiin vakiotietueita ja vakiokuormituksia A (50/50 luku/päivitys), B (95/5 luku/päivitys), C (100 % luku), G (5/95 luku/päivitys) ja H (100 % päivitys). Kuormitukset ajettiin kolmelle eri kokoiselle tietokannalle ja mitattiin useamman suorituskerran keskiarvo. Taulukoissa 2 ja 3 on esitetty mittaustulokset lukupainotteisilla kuormituksilla B ja C.

	Tietokannan koko (tietuetta)		
Tietokanta	**100 000**	**280 000**	**700 000**
MongoDB	12 s	22 s	32 s
Cassandra	29 s	21 s	18 s

Taulukko 2: MongoDB:n ja Cassandran suoritusaikoja kuormituksella B (95/5) [Abramova and Bernardino 2013]

	Tietokannan koko (tietuetta)		
Tietokanta	**100 000**	**280 000**	**700 000**
MongoDB	16 s	27 s	35 s
Cassandra	43 s	24 s	20 s

Taulukko 3: MongoDB:n ja Cassandran suoritusaikoja kuormituksella C (100 % luku) [Abramova and Bernardino 2013]

Molemmilla kuormituksilla MongoDB:n suoritusaika oli lyhyempi vain silloin, kun tietokannan koko oli 100 000. Tietokannan koon kasvaessa sen suoritusaika alkoi kasvaa. Sen sijaan Cassandran suoritusaika lyheni mitä suurempi oli tietokannan koko.

Päivityspainotteisen kuormituksen A (50/50) tulokset on esitetty taulukossa 4. Cassandran suoritusaika on pienempi riippumatta tietokannan koosta. Tietokannan koolla ei myöskään ole juuri sen vaikutusta suoritusaikaan. MongoDB:n suoritusaika kasvoi tietokannan kasvaessa 100 000 -> 280 000, mutta laski hieman välillä 280 000 -> 700 000. Kuormituksella A molempien tietokantojen suoritusaika oli lyhyempi 700 000:n tietueen tietokannassa kuin 280 000:en. Tutkijoiden mukaan tämä johtuu siitä, että tässä kohtaa tietuemäärä on kasvanut niin suureksi, että tietokantojen optimointiominaisuudet tulevat käyttöön.

	Tietokannan koko (tietuetta)		
Tietokanta	**100 000**	**280 000**	**700 000**
MongoDB	19 s	31 s	28 s
Cassandra	10 s	14 s	11 s

Taulukko 4: Tietokantojen suoritusaikoja kuormituksella A (50/50) [Abramova and Bernardino 2013]

Taulukoissa 5 ja 6 on esitetty tulokset kirjoituspainotteisista kuormituksista G (5/95) ja H (100% kirjoitus). Molemmissa Cassandran suoritusaika oli erittäin lyhyt, eikä se muuttunut tietokannan koon kasvaessa. MongoDB:n suoritusaika oli huomattavasti pidempi ja kasvoi tietokannan koon kasvaessa, erityisen paljon pelkistä kirjoitusoperaatioista koostuvalla kuormituksella G.

	Tietokannan koko (tietuetta)		
Tietokanta	**100 000**	**280 000**	**700 000**
MongoDB	23 s	31 s	36 s
Cassandra	1 s	2 s	3 s

Taulukko 5: Tietokantojen suoritusaikoja kuormituksella G (95/5) [Abramova and Bernardino 2013]

	Tietokannan koko (tietuetta)		
Tietokanta	**100 000**	**280 000**	**700 000**
MongoDB	25 s	27 s	43 s
Cassandra	1 s	1 s	1 s

Taulukko 6: MongoDB:n ja Cassandran suoritusaikoja kuormituksella H (100 % kirjoitus) [Abramova and Bernardino 2013]

Useamman NoSQL-tietokannan suorituskykyä vertailtiin Abramovan ja muiden [2014a] tutkimuksessa. Tutkimuksessa käytettiin YCSB:n vakiokuormituksia A (50/50), B (95/5), C (100 % luku), G (5/95) ja H (100 % kirjoitus). Vertailtavat tietokannat olivat avain-arvoparitietokannat Tarantool, Redis, Voldemort, Scalaris ja Oracle NoSQL, dokumenttitietokannat MongoDB, Elasticsearch ja OrientDB sekä sarakeperheet HBase ja

Cassandra. Tietokantoihin tallennettiin ensin 600 000 tietuetta, minkä jälkeen ajettiin 1000 kuormituksen mukaista tietokantaoperaatiota. Tulokset kuormituksella B (95/5) on esitetty taulukossa 7. Selvästi lyhyimmät suoritusajat olivat avain-arvoparitietokannoilla Tarantool, Redis ja Scalaris. Dokumenttitietokannoista Elasticsearch oli nopein. Hitaimmat olivat HBase ja OrientDB.

Tietokanta	**Tyyppi**	**Suoritusaika (s)**
Tarantool	Avain-arvopari	0,40
Redis	Avain-arvopari	0,51
Scalaris	Avain-arvopari	0,99
Elasticsearch	Dokumentti	3,81
Oracle NoSQL	Avain-arvopari	14,42
MongoDB	Dokumentti	15,34
Cassandra	Sarakeperhe	15,67
Voldemort	Avain-arvopari	16,11
HBase	Sarakeperhe	22,32
OrientDB	Dokumentti	30,09

Taulukko 7: Suoritusaikoja kuormituksella B (95/5) [Abramova *et al.* 2014a]

Tulokset kuormituksella C (100% luku) on esitetty taulukossa 8. Lyhyimmät suoritusajat olivat tässäkin avain-arvoparitietokannoilla Tarantool, Redis ja Scalaris, joiden suoritusaika oli alle sekunnin. Muut avain-arvoparitietokannat Voldemort ja Oracle NoSQL olivat huomattavasti näitä hitaampia. Sarakeperheistä Cassandra oli HBasea nopeampi. Dokumenttitietokannoista Elasticsearch oli MongoDB:a ja OrientDB:a nopeampi.

Tietokanta	**Tyyppi**	**Suoritusaika (s)**
Tarantool	Avain-arvopari	0,33
Redis	Avain-arvopari	0,49
Scalaris	Avain-arvopari	0,89
Elasticsearch	Dokumentti	3,74
Cassandra	Sarakeperhe	15,07
Oracle NoSQL	Avain-arvopari	15,17
Voldemort	Avain-arvopari	16,28
MongoDB	Dokumentti	16,00
OrientDB	Dokumentti	21,81
HBase	Sarakeperhe	28,22

Taulukko 8: Suoritusaikoja kuormituksella C (100 % luku) [Abramova *et al*. 2014a]

Myös päivityspainotteisella kuormituksella A (50/50) avain-arvoparitietokannat olivat yleisesti ottaen muita nopeampia. Tulokset nopeusjärjestyksessä on esitetty taulu-

kossa 9. Erityisen nopeita olivat Tarantool ja Redis. Voldemort oli muita hitaampi. Dokumenttitietokantojen välillä oli suuria eroja. Niistä parhaiten suoriutui Elasticsearch. OrientDB oli selvästi muita hitaampi. Sarakeperheistä Cassandra oli HBasea nopeampi.

Tietokanta	Tyyppi	Suoritusaika (s)
Tarantool	Avain-arvopari	0,44
Redis	Avain-arvopari	0,54
Scalaris	Avain-arvopari	1,65
Elasticsearch	Dokumentti	3,83
Cassandra	Sarakeperhe	7,89
Voldemort	Avain-arvopari	13,27
MongoDB	Dokumentti	17,10
HBase	Sarakeperhe	21,35
OrientDB	Dokumentti	30,09

Taulukko 9: Suoritusaikoja kuormituksella A (50/50) [Abramova *et al.* 2014a]

Taulukossa 10 on esitetty kuormituksen G (5/95) tulokset. Tuloksista voidaan nähdä, että avain-arvoparitietokantojen suoritusajat olivat hyvin lyhyitä Voldemortia lukuun ottamatta. Enemmän lukuoperaatioita sisältävään kuormitukseen A (50/50) verrattuna sarakeperheiden suorituskyky oli tässä huomattavasti parempi. Dokumenttitietokannoilla ei ollut suurta eroa näiden kahden kuormituksen välillä.

Tietokanta	Tyyppi	Suoritusaika (s)
Tarantool	Avain-arvopari	0,44
Redis	Avain-arvopari	0,49
Cassandra	Sarakeperhe	1,58
Scalaris	Avain-arvopari	2,30
Elasticsearch	Dokumentti	3,77
HBase	Sarakeperhe	4,31
Oracle NoSQL	Avain-arvopari	5,55
Voldemort	Avain-arvopari	14,15
MongoDB	Dokumentti	18,65
OrientDB	Dokumentti	36,26

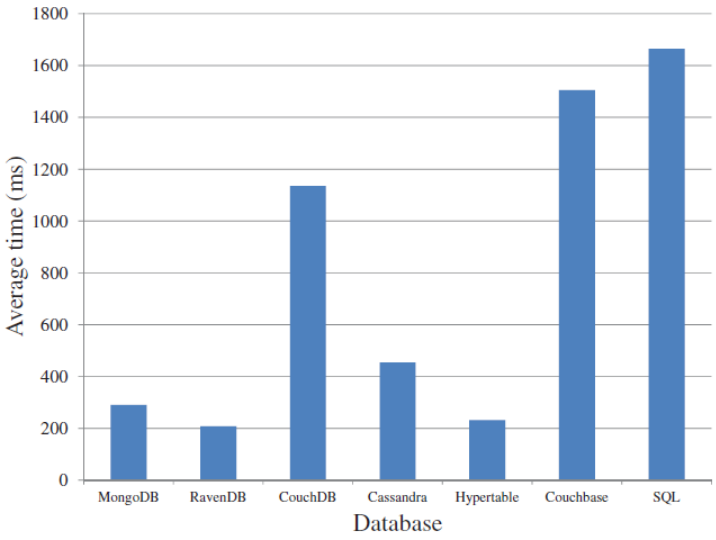
Taulukko 10: Suoritusaikoja kuormituksella G (5/95) [Abramova *et al*. 2014a]

Hyvin samanlaiset tulokset saatiin kuormituksella H (100 % kirjoitus). Tämän kuormituksen käsittelyn suoritusajat on esitetty taulukossa 11. Kuormituksen ollessa pelkkiä kirjoitusoperaatioita sarakeperheiden nopeus parani entisestään edelliseen kuormitukseen verrattuna. Tutkijoiden mukaan Cassandran hyvä, HBasea parempi suorituskyky kirjoitusoperaatioissa johtuu sen käyttämästä välimuistilokista. Kaikki tapahtumat kirjoitetaan lokiin ja vain lokin sisältö viedään levylle tiedostojen jäädessä välimuistiin, mikä vähentää levyoperaatioita. OrientDB:n suorituskyky oli jälleen huonoin.

Tietokanta	Tyyppi	Suoritusaika (s)
Redis	Avain-arvopari	0,49
Tarantool	Avain-arvopari	0,57
Cassandra	Sarakeperhe	0,97
HBase	Sarakeperhe	1,29
Scalaris	Avain-arvopari	2,38
Elasticsearch	Dokumentti	3,83
Oracle NoSQL	Avain-arvopari	4,49
Voldemort	Avain-arvopari	13,20
MongoDB	Dokumentti	21,68
OrientDB	Dokumentti	36,75

Taulukko 11: Suoritusaikoja kuormituksella H (100 % kirjoitus) [Abramova *et al.* 2014a]

Li ja Manoharan [2013] vertailivat useiden NoSQL-tietokantojen ja relaatiotietokannan suorituskykyä. Tutkimuksen tietokannat olivat dokumenttitietokannat MongoDB, RavenDB ja CouchDB, avain-arvopari -tietokannan ja dokumenttitietokannan hybridi Couchbase, sarakeperheet Cassandra ja Hypertable sekä relaatiotietokanta Microsoft SQL Server Express. Tutkimuksessa ei käytetty YCSB:ia. Tarvittavat luonti-, luku-, kirjoitus- ja poisto-operaatiot toteutettiin Java-kielisellä ohjelmalla. Tutkimusdatana oli eri kokoisia, yksinkertaisia avain-arvopareja sisältäviä tiedostoja. Mittauksia tehtiin myös hyvin pienillä tietuemäärillä (10, 50, 100, 1000). Nämä tulokset jätetään tässä käsittelemättä ja esitellään tulokset vain tietuemäärällä 100 000. Luontioperaatiossa luotiin tietovaraston ilmentymä. Testi toistettiin viisi kertaa ja laskettiin operaation suorittamiseen kulunut keskimääräinen aika. Tulokset on esitetty kuvassa 10.



Kuva 10: Keskimääräinen suoritusaika tietovaraston ilmentymän luonnissa [Li and Manoharan 2013]

Lyhyimmät suoritusajat olivat RavenDB:lla, Hypertablella ja MongoDB:lla. Näitä hieman hitaampi oli Cassandra. Selvästi muita hitaampia olivat CouchDB, Couchbase ja MS SQL Server Express.

Suorituskykyä lukuoperaatiossa testattiin hakemalla kaikki tietokannan tietueet annetun avaimen perusteella. Testitulokset on esitetty taulukossa 12. Tietokantojen järjestys nopeimmasta hitaimpaan oli Couchbase, MongoDB, MS SQL Express, Hypertable, CouchDB, Cassandra ja RavenDB. Erot suoritusajoissa olivat erittäin suuria erityisesti dokumenttitietokantojen välillä.

Tietokanta	Tyyppi	Suoritusaika (s)
Couchbase	Dokumentti	7 s
MongoDB	Dokumentti	10 s
MS SQL Express	Relaatiotietokanta	17 s
Hypertable	Sarakeperhe	63 s
CouchDB	Dokumentti	176 s
Cassandra	Sarakeperhe	228 s
RavenDB	Dokumentti	426 s

Taulukko 12: Suoritusaikoja pelkissä lukuoperaatioissa [Li and Manoharan 2013]

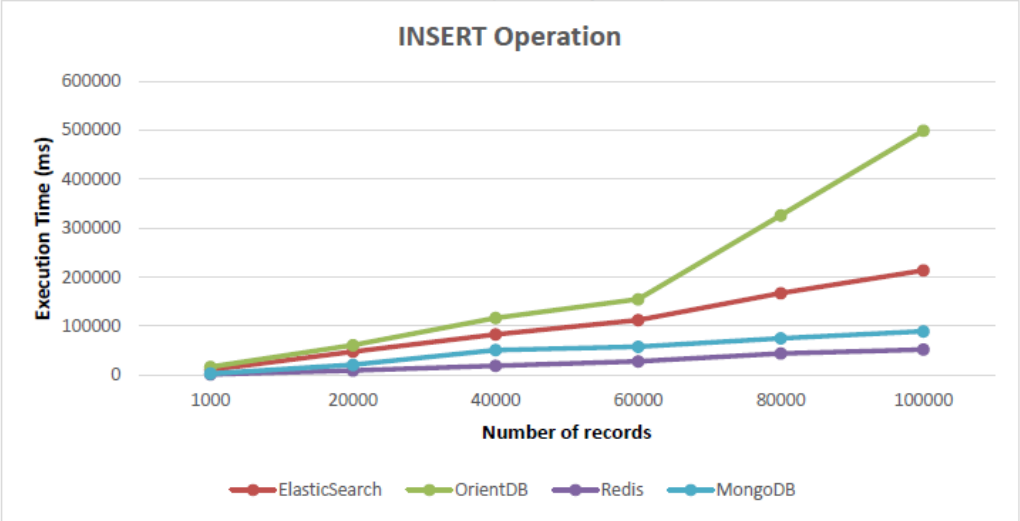
Kolmannessa testissä tutkittiin suoritusaikaa pelkissä kirjoitusoperaatioissa. Tietokantaan kirjoitettiin uusi tietue, mikäli sitä ei vielä löytynyt ja muussa tapauksessa päivitettiin olemassa olevaa tietuetta. Tulokset on esitetty taulukossa 13. Järjestys lyhyimmästä suoritusajasta pisimpään oli Couchbase, MongoDB, Cassandra, Hypertable, MS SQL Server Express, RavenDB ja CouchDB. Tietokantojen välillä oli hyvin suuria eroja. RavenDB:n ja CouchDB:n suoritusajat oli huomattavan pitkiä muihin verrattuna. Sen sijaan Couchbasen suoritusaika oli erittäin lyhyt.

Tietokanta	Tyyppi	Suoritusaika (s)
Couchbase	Dokumentti	8 s
MongoDB	Dokumentti	23 s
Cassandra	Sarakeperhe	88 s
Hypertable	Sarakeperhe	115 s
MS SQL Express	Relaatiotietokanta	216 s
RavenDB	Dokumentti	740 s
CouchDB	Dokumentti	932 s

Taulukko 13: Suoritusaikoja pelkissä kirjoitusoperaatioissa [Li and Manoharan 2013]

Abubakarin ja muiden [2014] tutkimuksessa testattiin dokumenttitietokantojen Elasticsearch, MongoDB, OrientDB ja avain-arvoparitietokanta Rediksen suorituskykyä. Tutkimuksessa käytettiin YCSB:ia ja sen vakiokuormituksia ja -tiedostoja. Tietokantaa

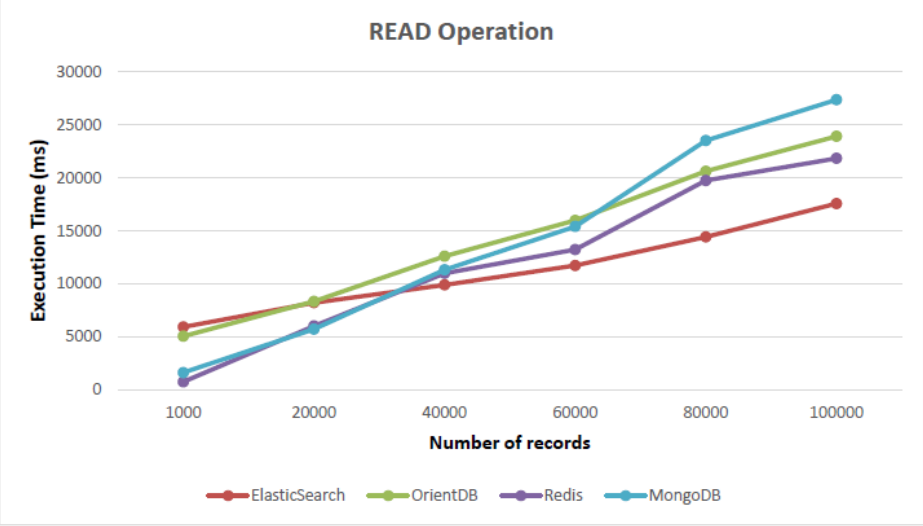
kuormitti kymmenen samanaikaista asiakasta. Testattavat kuormitukset olivat pelkät lisäysoperaatiot, pelkät lukuoperaatiot ja pelkät päivitysoperaatiot. Kuhunkin tietokantaan ladattiin ensin tietuemäärät 1000, 20 000, 40 000, 80 000 ja 100 000 ja mitattiin siihen kulunut aika. Mittaustulokset on esitetty kuvassa 11.



Kuva 11: Tietokantojen suoritusaikoja latausoperaatioissa [Abubakar *et al.* 2014]

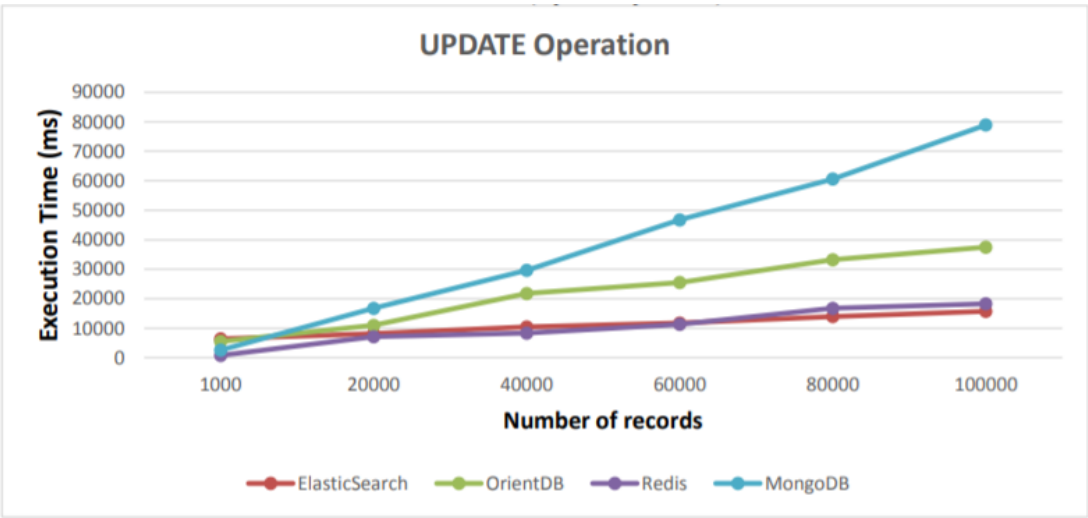
Tietokannan koosta riippumatta Rediksen suoritusaika oli lyhyin. Toiseksi parhaiten suoriutuivat MongoDB ja Elasticsearch. Pisin suoritusaika oli OrientDB:lla. Lisäksi sen suoritusaika alkoi kasvaa muita jyrkemmin tietuemäärän kasvaessa yli 60 000:en. Tutkijoiden mukaan Rediksen suoritusaika oli lyhyt, koska se on kirjoitusoptimoitu ja suorittaa operaatiot muistissa. OrientDB:n suorituskykyyn vaikuttanee sen tietomalli, joka on osaksi graafitietomalli.

Suorituskykyä lukuoperaatioissa testattiin lukemalla kaikki tietokannan tietueet. Kuvassa 12 esitetyistä tuloksista voidaan nähdä, että pienillä tietuemäärillä nopeimmat olivat Redis ja MongoDB. Tietuemäärän kasvaessa yli neljänkymmenentuhannen, Elasticsearchin suoritusaika oli pienin. Tietuemäärän ollessa 100 000, järjestys nopeimmasta hitaimpaan oli Elasticsearch, Redis, OrientDB ja MongoDB. Kaikilla tietokannoilla suoritusaika kasvoi tietuemäärän kasvaessa eikä suoritusajoissa ylipäätään ollut suuria eroja.



Kuva 12: Tietokantojen suoritusaikoja lukuoperaatioissa [Abubakar *et al.* 2014]

Tietokantojen suoritusaikoja kirjoitusoperaatioissa mitattiin päivittämällä kaikki tietokannan tietueet. Mittaustulokset on esitetty kuvassa 13. Tietokannan koon ollessa pieni (1000), ei suoritusajoissa ollut juuri eroja. MongoDB:n suoritusaika alkoi kasvaa muita nopeammin tietuemäärän kasvaessa. Redisin ja Elasticsearchin suoritusaika pysyi lähes vakiona tietuemäärästä riippumatta. Sadantuhannen tietueen tietokannalla järjestys suoritusajan mukaan pienimmästä suurimpaan oli Elasticsearch, Redis, OrientDB ja MongoDB.



Kuva 13: Tietokantojen suoritusaikoja päivitysoperaatioissa [Abubakar *et al.* 2014]

Tangin ja Fanin [2016] tutkimuksessa vertailtiin Rediksen, MongoDB:n, Couchbasen, HBasen ja Cassandran suorituskykyä. Jokainen tietokanta hajautettiin neljän palvelimen klusterille. Yhdelle palvelimelle asennettiin YCSB. Tutkimuksessa käytettiin

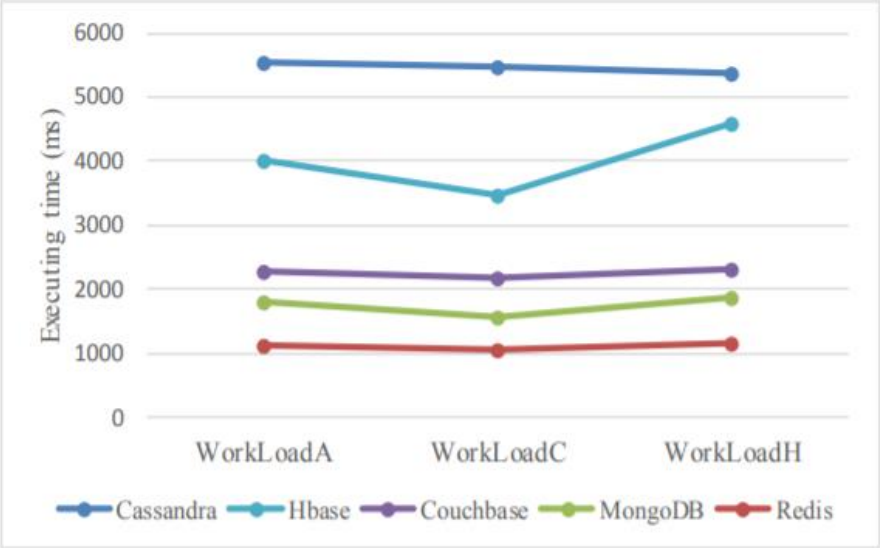
YCSB:n vakiokuormituksia A (50/50), C (100% luku) ja H (100% kirjoitus). Ensimmäisessä vaiheessa mitattiin suoritusaika, kun tyhjään tietokantaan kirjoitettiin 100 000 tietuetta. Suoritusajat on esitetty taulukossa 14.

Tietokanta	Tyyppi	Suoritusaika (s)
Redis	Avain-arvopari	8 s
MongoDB	Dokumentti	10 s
Cassandra	Sarakeperhe	14 s
HBase	Sarakeperhe	15 s
Couchbase	Dokumentti	20 s

Taulukko 14: Suoritusnopeudet latausoperaatiossa [Tang and Fan 2016]

Järjestys nopeimmasta hitaimpaan oli Redis, MongoDB, Cassandra, HBase ja Couchbase. Tutkijoiden mukaan Rediksen nopeus johtuu sen käyttämästä puolipysyvästä tilasta. Cassandran ja HBasen nopeutta kirjoitusoperaatioissa lisää se, että tiedot kirjoitetaan ensin välimuistissa sijaitsevaan lokiin. Operaatio katsotaan tässä vaiheessa onnistuneesti suoritetuksi. Couchbasessa tiedon kirjoittamisessa tietokantaan on useampia vaiheita ja mekanismeja, mikä teki siitä testin hitaimman.

Suorituskykyä erilaisilla kuormituksilla testattiin ajamalla 10 000 tietokantaoperaatiota testiä kohden. Tulokset on esitetty kuvassa 14.

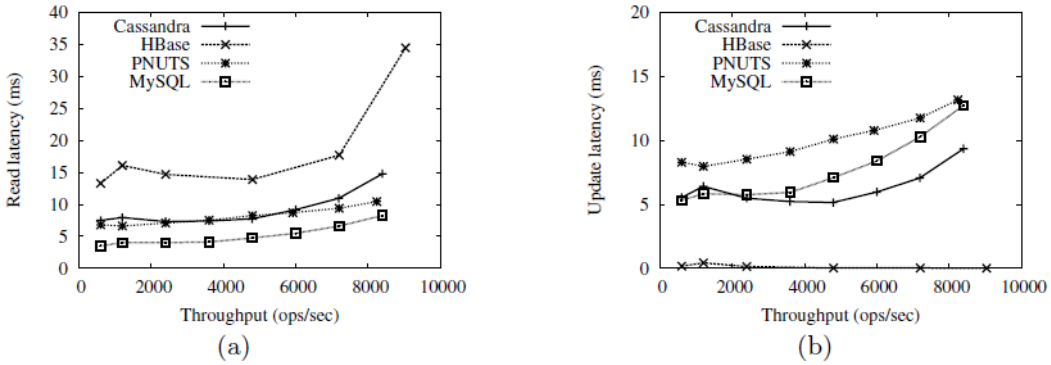


Kuva 14: Suoritusaikoja kuormituksilla A (50/50), C (100 % luku) ja H (100 % kirjoitus) [Tang and Fan 2016]

Kuormituksesta riippumatta Rediksen suoritusaika oli lyhyin. Seuraavina olivat MongoDB ja Couchbase. MongoDB:n suoritusaikaa parantavat sen käyttämät asynkroniset luku- ja kirjoitusoperaatiot. Couchbasessa käytetään muistivarastoarkkitehtuuria, jossa

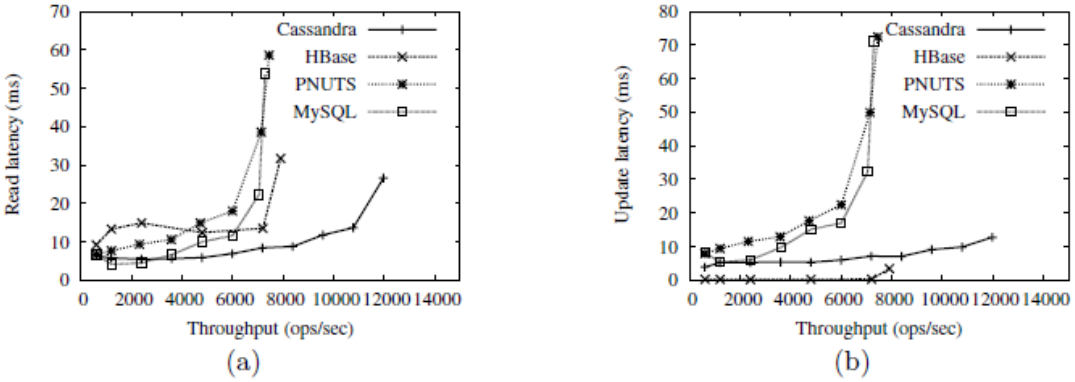
käydyt dokumentit ja indeksit pidetään välimuistissa sijaitsevassa hajautustaulussa. Tämä parantaa tiedon löydettävyyttä sekä pienentää luku- ja kirjoitusviivettä. HBase ja Cassandra olivat testin hitaimmat. Muista tutkimuksista poiketen HBase oli tässä Cassandraa nopeampi. Tämä johtui tutkijoiden mukaan HBasen uudesta versiosta, jossa on tehty lukuoptimointia BlockCache-tekniikan avulla. Siinä välimuistiin viedään viimeksi luetut tulokset ja vanhin data eliminoidaan.

Cooper ja muut [2010] tutkivat sarakeperheiden Cassandra ja HBase, PNUTS:in ja relaatiotietokanta MySQL:n suorituskykyä erittäin suurilla tietuemäärillä. Tietokannat hajautettiin kuudelle pilvipalvelussa sijaitsevalle palvelimelle. Testauksessa käytettiin YCSB:ia ja sen vakiotietueita ja -kuormituksia. Tietuemäärä oli 120 miljoonaa. Tutkimuksessa tarkasteltiin suoritustehoa suhteessa viiveeseen. Kuvassa 15 on esitetty mittaustulokset kuormituksella B (95/5). YCSB antaa mittaustulokset erikseen luku- ja kirjoitusviiveen osalta. Kuvan kohdassa esitetty tulokset erikseen luku- ja kirjoitusviiveen osalta.



Kuva 15: Viive ja suoritusteho kuormituksella B (95/5) [Cooper *et al.* 2010]

MySQL:n lukuviive oli pienin. Cassandran ja PNUTSin tulokset olivat lähes samanlaiset. HBasen lukuviive oli muita suurempi ja se kasvoi yhtäkkiä hyvin suureksi suoritustehon kasvaessa yli 8000:een operaatioon sekunnissa. Sen sijaan sen kirjoitusviive oli erittäin pieni ja selvästi muita pienempi. Tutkijoiden mukaan lukuviivettä aiheuttaa HBasessa se, että tietueiden osia haetaan useista tiedostoista levyltä ja tietueet uudelleenrakennetaan löydetyistä osista. Myös Cassandra kokoaa tietueet luettaviksi levyllä, mikä aiheuttaa ylimääräisiä levytapahtumia. Vaikutus on tosin huomattavissa vasta hyvin suurilla tietuemäärillä, silloin kun tietokanta on lähellä saturaatiota. Suorituskykyä testattiin myös päivityspainotteisella kuormituksella A (50/50). Mittaustulokset on esitetty kuvassa 16.



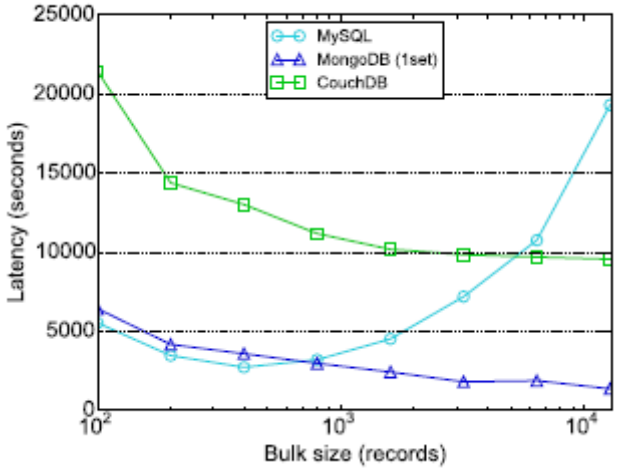
Kuva 16: Viive ja suoritusteho kuormituksella A (50/50) [Cooper *et al.* 2010]

Lukuoperaatioiden osalta Cassandran viive pysyi pienimpänä suoritustehon kasvaessa ja sen maksimisuoritusteho oli muita parempi. Päivitysoperaatioissa HBasen viive oli pienin. MySQL:n ja PNUTS:n viive oli selvästi sarakeperheitä suurempi.

7.2 Suorituskyky tapaustutkimuksissa

Eräs NoSQL-tietokantojen tyypillisimmistä käyttökohteista on IoT-datan käsittely. IoT-data on usein tyypiltään yksinkertaista sensoridataa. Phan ja muut [2014] tutkivat dokumenttitietokantojen MongoDB ja CouchDB, avain-arvoparitietokanta Rediksen ja relaatiotietokanta MySQL:n suorituskykyä sensoridatan kirjoittamisessa ja lukemisessa. Tutkimuksessa simuloitiin järjestelmää, jossa yksilöivästi tunnistettavat sensorisolmut tuottavat skalaaridataa ja generoivat yhden lukeman kerran intervallin aikana. Sensorisolmuja simuloitiin datageneraattorin avulla. Generaattori muodosti satunnaisia arvoja sisältäviä tietuelistoja. Lista välitettiin pilvipalvelussa sijaitsevalle tietokantapalvelimelle lähettäjän (*data sender*) avulla.

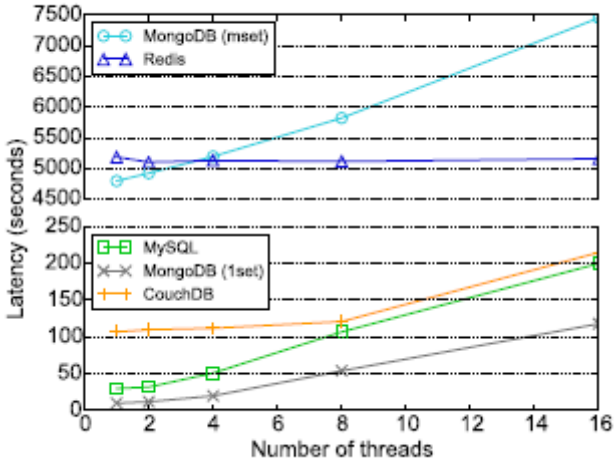
Massalisäyksellä (*bulk insert*) tarkoitetaan useamman tietueen kirjoittamista tietokantaan yhdellä tietokantaoperaatiolla [Phan *et al.* 2014]. Massalisäys on mahdollista vain silloin, kun tietokanta koostuu yhdestä taulusta tai kokoelmasta. Phanin ja muiden [2014] tutkimuksessa vertailtiin massalisäyksen vaikutusta suorituskykyyn ja tietokannan kokoon. MongoDB:ssä käytettiin yhden (1set) ratkaisua, CouchDB:ssä yhtä kokoelmaa ja MySQL:ssa yhtä taulua. Rediksessä tietorakenne koostuu useista hajautuksista, joten massalisäys ei ole mahdollista. Testissä tietokantoihin tallennettiin erikokoisia listoja. Testi toistettiin vähintään kymmenen kertaa ja mitattiin keskimääräinen viive. Tulokset on esitetty kuvassa 17, josta nähdään viive sekunteina listan koon kasvaessa.



Kuva 17: Tietokantojen viive massalisäyksessä [Phan *et al*. 2014]

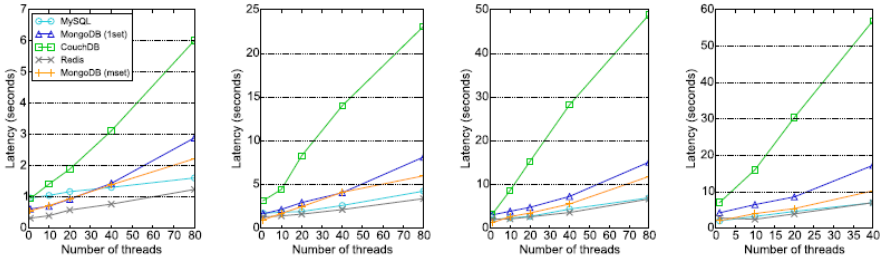
MongoDB:n viive oli lyhyin. Lisäksi viive lyheni edelleen tietuemäärän kasvaessa. CouchDB oli selvästi hitaampi, mutta myös sen viive lyheni. MySQL:n viive oli lähes yhtä pieni kuin MongoDB:n, mutta sen viive alkoi kasvaa tietuemäärän kasvaessa. Massalisäyksen havaittiin olevan yksittäisten tietueiden lisäämistä nopeampaa. Massalisäys voi vaikuttaa merkittävästi myös tietokannan kokoon. CouchDB:n BTree-tietorakenteen takia massalisäys tuotti 14 kertaa pienemmän tietokannan kuin tietueiden lisääminen yksittäin.

Massalisäyksen lisäksi tutkittiin suorituskykyä tilanteessa, kun tietokantaan kirjoittaa samanaikaisesti useampi käyttäjä. Käyttäjiä simuloitiin säikeiden avulla. Jokaista säiettä kohden oli yksi lähettäjä, joka vastasi tuhannen sensorisolmun ryhmästä. Joka kerta, kun ryhmä generoi tietueita, lähetettiin tietokantaan 1000 tietuetta. Lähetyskertoja oli 100, jolloin tietokantaan lisättiin yhteensä 100 000 tietuetta. Massalisäystä käytettiin soveltuville tietokannoille. MongoDB:n osalta testattiin yhden kokoelman lisäksi useamman kokoelman toteutusta (mset). Tulokset on esitetty kuvassa 18. Viive oli selvästi pienempi massalisäyksen mahdollistavilla tietokannoilla. Kaikkien tietokantojen viive kasvoi säikeiden määrän kasvaessa lukuun ottamatta Redistä, joka oli ainoa muistissa toimiva tietokanta.



Kuva 18: Tietokantojen viive säikeiden määrän funktiona [Phan *et al.* 2014]

Tietokantoja testattiin myös useamman samanaikaisen käyttäjän suorittaessa lukuoperaatioita. Testeissä käytettiin erilaisia säie- ja tietuemääriä. Tietuemäärät olivat 1, 10, 20 ja 40 miljoonaa. Jokainen säie luki vain tiettyä sensoria koskevaa dataa. Tulokset on esitetty kuvassa 19. Tietokantojen välillä ei ollut suuria eroja lukuun ottamatta CouchDB:ia, jonka viive oli muita suurempi kaikissa testeissä. Parhaat tulokset antoi Redis johtuen muistissa toimimisesta. MySQL oli lähes yhtä nopea silloin, kun tietueiden määrä oli 10 miljoonaa tai yli. MongoDB:ssä kokoelmien määrä vaikutti viiveeseen. Useammalla kokoelmalla viive oli lyhyempi kuin yhdellä.



Kuva 19: Tietokantojen viive lukuoperaatioissa [Phan *et al.* 2014]

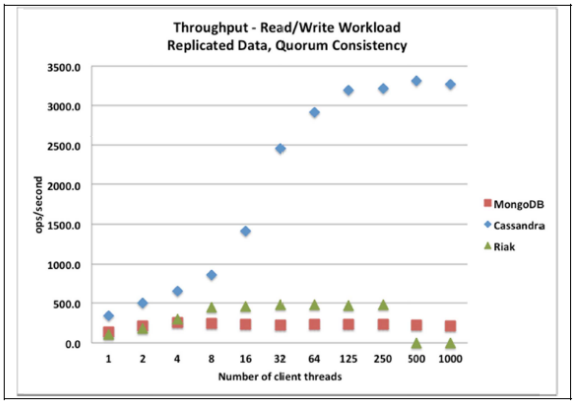
Tutkimuksen tuloksena saatiin tietoa tietokantojen suorituskyvystä sensoridatan käsittelyssä. Tulosten perusteella sekä NoSQL- että SQL-tietokannat soveltuvat tähän käyttöön. Kuitenkaan Redis ja CouchDB eivät olisi parhaat mahdolliset valinnat. Tietomalli vaikuttaa tietokannan viiveeseen eri tavoin luku- ja kirjoitusoperaatioiden suorittamisessa. Kirjoitusoperaatioiden suorittamisessa parhaimmillaan ovat tietokannat, joissa

massalisäys on mahdollista. Tällaisia ovat tietokannat, joissa data sijaitsee yhdessä kokoelmassa, hajautuksessa tai taulussa. Lukuoperaatioissa tilanne on päinvastainen ja useamman kokoelman tietokanta toimii nopeammin. Yksi tärkeimmistä asioista tietokannan valinnassa onkin määritellä, onko kuormitus pääasiassa joko kirjoitus- tai lukutyyppistä ja millainen on näiden suhde.

Yhtenä suurimmista NoSQL-tietokantojen eduista verrattuna relaatiotietokantoihin on niiden hyvä horisontaalinen skaalautuvuus. Näin ollen niiden käyttö hajautetuissa ratkaisuissa on luontevaa. Kleinin ja muiden [2015] tutkimuksessa tarkasteltiin mahdollisuuksia keskitetyn relaatiotietokannan korvaamiseen NoSQL-tietokannalla. Tutkimuksessa vertailtiin Cassandran, Riakin ja MongoDB:n suorituskykyä ja skaalautuvuutta sekä järjestelmän päätietokantana että sen välimuistina. Tutkimustapauksena oli suuren, maailmanlaajuisesti toimivan terveydenhuoltoalan yrityksen sähköisen potilastiedon hallinta. Sekä tietokantapalvelimet että asiakas toteutettiin Amazon EC2-pilvipalvelussa. Asiakaskoneelle asennettiin YCSB. Tietokannat hajautettiin kolmelle kolmen solmun klusterille. Vertailun vuoksi systeemi toteutettiin myös ilman hajautusta yhdellä tietokantapalvelimella.

Tutkimuksen data koostui miljoonasta potilastietueesta ja näihin yhden suhde moneen -tyyppisesti liittyvästä kymmenestä miljoonasta tutkimustulostietueesta. Tyypillisen kuormituksen arvioitiin olevan 80 % luku- ja 20 % kirjoitusoperaatioita. Lukuoperaatiossa luettiin potilaan viisi viimeisintä tulosta ja kirjoitusoperaatioissa kirjoitettiin tietokantaa yksi uusi tutkimustulostietue. Samanaikaisten käyttäjien määrä oli välillä 1 – 1000. Tutkimus suoritettiin käyttäen sekä vahvaa johdonmukaisuutta että lopulta johdonmukaisuutta. Vahvan johdonmukaisuuden havaittiin pienentävän tehoa 10 - 25 % tietokannasta riippuen. Tässä esitellään tulokset, jotka saatiin käytettäessä vahvaa johdonmukaisuutta.

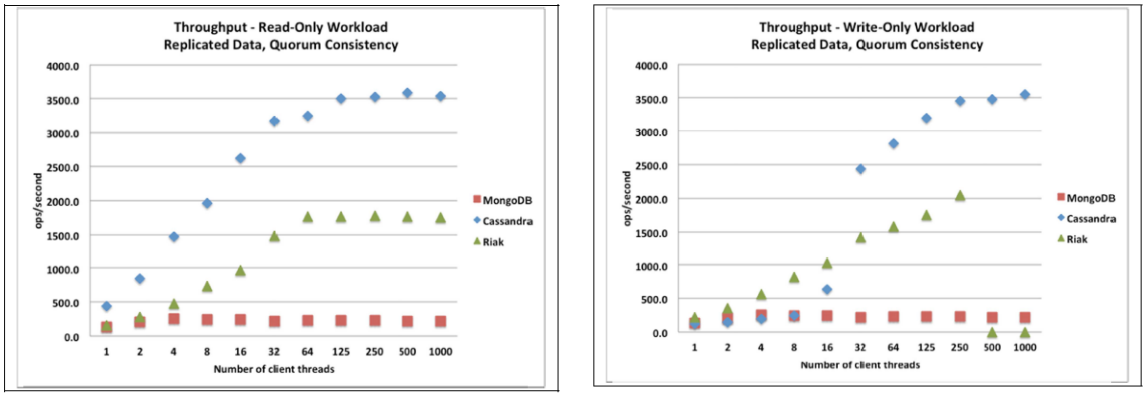
Tutkimuksen tuloksia on esitetty kuvassa 20. Cassandran suorituskyky oli hajautetussa ratkaisussa jopa hieman yhden solmun ratkaisua parempi. Cassandrassa asiakasyhteydet jakautuvat tasaisesti kaikille solmuille saman klusterin sisällä.



Kuva 20: Tietokantojen suoritustehoja 80/20 -kuormituksella [Klein *et al.* 2015]

Hajautuksen tuomat edut olivat suuremmat kuin solmujen välisen koordinoinnin hidastava vaikutus. Riakissa asiakasyhteydet jakautuvat tasaisesti kaikkien solmujen välille. Hajautuksen vaikutus suorituskykyyn oli Cassandraa suurempi, sillä suorituskyky oli hajauttamatonta neljä kertaa parempi. Suurin vaikutus hajautuksella oli MongoDB:n suorituskykyyn. Hajautetun ratkaisun suoritusteho oli vain 10 % yhden solmun ratkaisun suoritustehosta. Tämä johtui tutkijoiden mukaan isäntä-orja -arkkitehtuurista, jossa kaikki asiakasyhteydet kohdistuvat yhteen reititinsolmuun, jolloin se muodostuu tietokannan pullonkaulaksi. Reititinsolmu saavutti nopeasti saturaation, eikä yhtäaikaisten käyttäjien määrän kasvu siten vaikuttanut enää suoritustehoon. MongoDB:n suorituskykyyn vaikutti myös sen käyttämä uudelleentasapainottava, arvoväliin perustuva hajautus, mikä soveltui huonosti tutkimusdatan hajauttamiseen, sillä siinä uusille avaimille annettiin monotonisesti kasvava avain. Tämä johti siihen, että kaikki kirjoitusoperaatiot kohdistuivat samalle hajautukselle. Hajautusarvoihin perustuva hajautus tuli MongoDB:n versiossa 2.4.

NoSQL-tietokantaa voidaan käyttää myös välimuistina varsinaisen tietovaraston ja asiakkaan välillä. Ratkaisun avulla voidaan parantaa tiedon saatavuutta ja pienentää viivettä, sillä vain kulloinkin tarvittavat kirjoitus- ja lukutapahtumat käsitellään ensin välivarastossa. Tämän käyttötarkoituksen testaamisessa käytettiin kuormituksia, jotka koostuivat joko pelkistä luku- tai kirjoitusoperaatioista. Tulokset on esitetty kuvassa 21. Cassandran suoritusteho oli paras molemmilla kuormituksilla, mutta toisaalta myös sen viive oli suurin.



Kuva 21: Tietokantojen suoritustehoja säikeiden määrän funktiona [Klein *et al.* 2015]

Esimerkiksi silloin, kun samanaikaisia asiakkaita oli 32 kpl, Riakin lukuviive oli vain 20 % Cassandran viiveestä ja MongoDB:n kirjoitusviive oli 25 % Cassandran viiveestä. Näin ollen Riak ja MongoDB näyttäisivät soveltuvan välimuistiksi Cassandraa paremmin.

NoSQL-tietokannat on suunniteltu erityisesti valtavien tietomäärien käsittelyyn. Rabl ja muut [2012] tutkivat usean erilaisen tietokannan soveltuvuutta yrityksen sovellusten suorituskyvyn hallintajärjestelmän (APM, *Application Performance Management*) tietokannaksi. Tällainen järjestelmä tuottaa erittäin suuria tietomääriä. Testattavat tietokannat olivat avain-arvoparitietokannat Redis ja Voldemort, sarakeperheet Cassandra ja HBase, NewSQL-tietokanta VoltDB sekä relaatiotietokanta MySQL. Järjestelmän vaatimuksiin sopivaa dokumenttitietokantaa ei löydetty, joten ne jätettiin pois. APM-järjestelmän tarkoituksena on järjestelmän hallinnan lisäksi valvoa sovellusten tilaa ja tapahtumia [Rabl *et al.* 2012]. Järjestelmä voi tuottaa tietoa esimerkiksi tietyn palvelun vasteajasta tai tiettyjen palvelimien prosessorien käytöstä tietyllä ajanjaksolla. Tiedon avulla voidaan sekä analysoida järjestelmän nykytilaa että tehdä ennusteita tulevaisuudesta. Vaatimuksena tietokantajärjestelmälle oli järjestelmän tuottaman valtavan datamäärän hallinnan lisäksi mahdollisuus esittää reaaliaikaista tietoa tietystä kohteesta. Testauksessa käytettiin YCSB:ia. Tarvittavat kyselyt olivat yhden arvon luku sekä pienen arvovälin skannaus, jolloin aggregoidaan arvoja tietyltä aikaväliltä. Kuormitukset ja niiden operaatiojakaumat on esitetty taulukossa 15.

Kuormitus	**Luku**	**Skannaus**	**Lisäys**
R	95	0	5
RW	50	0	50
W	1	0	99
RS	47	47	6
RSW	25	25	50

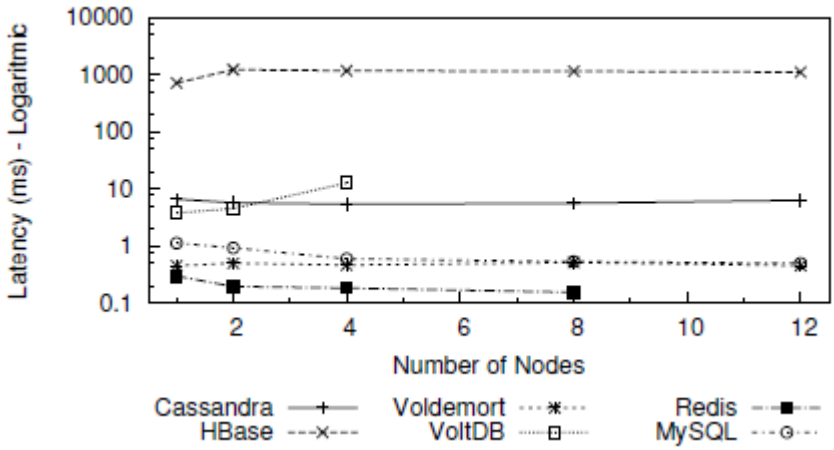
Taulukko 15: Tutkimuksessa käytetyt kuormitukset [Rabl *et al.* 2012]

Tietokannat hajautettiin kuudentoista solmun klusterille. Solmut sisälsivät kukin 10 miljoonaa tietuetta. Asiakasyhteyksiä oli 128 palvelinsolmua kohden. Jokaista testiä ajettiin 10 minuuttia ja mitattiin suoritusteho ja viive.

Kuormituksella R (95/5) lyhyimmät lukuviiveet olivat Rediksellä ja Voldemortilla. Seuraavaksi nopein oli MySQL. Cassandran viive selvästi pidempi ja HBase oli huomattavasti kaikkia muita hitaampi. Voldemortin, Cassandran ja HBasen viive kasvoi hieman, kun solmujen määrä kasvoi yhdestä kahteen, mutta pysyi sen jälkeen vakiona. Redisin ja MySQL:n viive sen sijaan laski solmujen määrän kasvaessa. VoltDB:n viive oli keskinkertainen mutta kasvoi jyrkästi kahden ja neljän solmun välillä. Kirjoitusoperaatioiden osalta tietokantojen erot olivat hieman suuremmat. HBasen viive oli pienin. Cassandran viive taas oli yllättäen suurin, sillä se on kirjoitusoptimoitu tietokanta. Voldemortin kirjoitusviive oli yhtä suuri kuin lukuviive. Redis oli MySQL:ää parempi. VoltDB:n viive oli suuri, eikä sitä saatu mitattu yli neljän solmun tilanteessa.

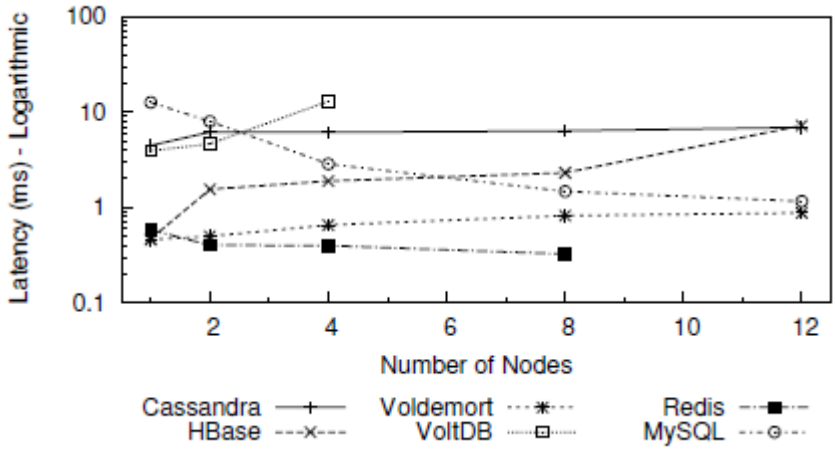
Kuormituksella RW (50/50) viiveet olivat lähes samat kuin kuormituksella R. Poikkeuksena oli MySQL, jonka lukuviive oli edellistä pienempi sekä yhdellä että kahdellatoista solmulla ja kirjoitusviive puolet suurempi. HBasen kirjoitusviive oli puolet pienempi kuin kuormituksella R.

Kuormitus W, joka koostui lähes pelkistä kirjoitusoperaatioista (99 % kirjoitus) vastasi parhaiten käyttökohteen tarpeita. Tietokantojen lukuviive tällä kuormituksella on esitetty kuvassa 22. HBase:n lukuviive oli huomattavan suuri ja oli keskimäärin 1 s kahdellatoista solmulla. Muiden lukuviive oli sitä selvästi pienempi. Rediksen lukuviive oli pienin. Voldemortin ja MySQL:n viiveet olivat lähes samat ja Cassandra oli näitä hitaampi. VoltDB:n viive oli melko suuri ja se kasvoi jälleen voimakkaasti solmujen määrän kasvaessa yhdestä neljään.



Kuva 22: Lukuviive kuormituksella W [Rabl *et al.* 2012]

Kirjoitusviive kuormituksella W on esitetty kuvassa 23. HBasen kirjoitusviive parani 20-kertaisesti kuormituksen RW verrattuna. Se oli selvästi sitä parempi, mitä suurempi osa kuormituksesta oli kirjoitusoperaatioita. Voldemortin kirjoitusviiveeseen tällä ei ollut vaikutusta. Muiden tietokantojen viive taas kasvoi 5 – 15 % edelliseen verrattuna.



Kuva 23: Kirjoitusviive kuormituksella W [Rabl *et al.* 2012]

8 Yhteenveto

Tutkimuksen tulokset esitetään tässä luvussa sekä tietomalleittain että tietokantakohtaisesti taulukoituna. Taulukoissa on esitetty tietokannan suorituskyky ja soveltuvuus erikseen luku- ja kirjoituspainotteisten kuormitusten osalta. Taulukossa on lisäksi muita huomioita, kuten hajautuksen ja samanaikaisten käyttäjien vaikutus suorituskykyyn. Lopuksi esitellään tietokantojen suorituskykyyn vaikuttavia tekijöitä.

Avain-arvoparitietokannat ovat yleisesti ottaen erittäin nopeita sekä luku- että kirjoitusoperaatioissa, sillä ne toimivat yleensä muistissa. Muistissa toimiminen rajoittaa toisaalta käsiteltävää tietomäärää. Yksinkertainen tietomalli tekee yksinkertaisten, avaimeen perustuvien operaatioiden suorittamisesta nopeaa. Avain-arvoparitietokannat sopivat hyvin erityisesti välimuistiksi. Tulokset on esitetty taulukossa 16. Tutkimuksissa käsiteltiin kuuden eri avain-arvoparitietokannan suorituskykyä. Näistä neljää oli käsitelty vain yhdessä tutkimuksessa.

Tietokanta	**Lukupainotteiset kuormitukset** B (95/5), C (100), 80/20	**Päivityspainotteiset kuormitukset** A (50/50), G (5/95), H (100)	**Muita huomioita** hajautus, samanaikaiset käyttäjät, skaalautuvuus
Oracle NoSQL	Melko hyvä, kun tietokannan koko kohtalainen.	Nopea, kun tietokannan koko kohtalainen.	Vain yhdessä tutkimuksessa.
Redis	Nopea tai erittäin nopea, kun tietuemäärä pienehkö tai kohtalainen. Nopea luku, koska toimii muistissa.	Nopea tai erittäin nopea, kun tietuemäärä kohtalainen. Nopeutta parantaa puolipysyvä tila. Massalisäyksen käyttö ei mahdollista, joten ei parhaimmillaan, kun suuri määrä kirjoitusoperaatioita.	Samanaikaisten käyttäjien määrällä ei juuri vaikutusta suorituskykyyn. Nopea itsenäisenä. Hajautuskirjasto voi heikentää suorituskykyä. Ei paras valinta kirjoituspainotteisen sensoridatan käsittelyyn.
Riak	Hyvä teho ja lyhyt suoritusaika. Hajautus parantaa suoritustehoa lukuoperaatioissa.	Hyvä teho.	Sopii välimuistiksi. Vain yhdessä tutkimuksessa.
Scalaris	Erittäin nopea kohtalaisella tietuemäärällä.	Nopea.	Vain yhdessä tutkimuksessa.
Tarantool	Lyhyt suoritusaika kohtalaisella tietuemäärällä, koska tietojen sijoittelu muistiin.	Nopea tai erittäin nopea kohtalaisella tietuemäärällä.	Vain yhdessä tutkimuksessa.
Voldemort	Keskiverto.	Keskiverto.	Lähes lineaarinen skaalautuvuus. Kirjoitus- ja lukuviive samansuuruiset ja vakaat käyttäjien määrän kasvaessa. Sopii käyttökohteisiin, jossa luku- ja kirjoitusoperaatioita yhtä paljon.

Taulukko 16: Avain-arvoparitietokantojen tulokset

Dokumenttitietokannat suoriutuivat paremmin luku- kuin kirjoituspainotteisista kuormituksista, mikä johtunee tietokannan hierarkkisesta rakenteesta. Dokumenttitietokannan suorituskykyyn kirjoitusoperaatioissa vaikuttaa huomattavasti se, sijaitsevatko dokumentit yhdessä vai useammassa kokoelmassa. Yhdellä kokoelmalla kirjoitus on nopeaa, mutta lukuoperaatioiden suorittaminen taas hitaampaa. Dokumenttien pakkaaminen parantaa suorituskykyä. Dokumenttitietokantojen etuina ovat monipuoliset kyselymahdollisuudet sekä tietorakenne, joka mahdollista avain-arvoparitietokantoja monimutkaisemman tiedon tallentamisen ja indeksoinnin. Tutkimuksen tulokset dokumenttitietokantojen osalta on esitetty taulukossa 17.

Tietokanta	**Lukupainotteiset kuormitukset** B (95/5), C (100), 80/20	**Päivityspainotteiset kuormitukset** A (50/50), G (5/95), H (100)	**Muita huomioita** hajautus, samanaikaiset käyttäjät, skaalautuvuus
Couchbase	Erittäin nopea.	Nopea.	Useita vaiheita kirjoitusoperaatioissa. Muistivarastoarkkitehtuuri pienentää viivettä.
CouchDB	Hidas, kun paljon yhtäaikaisia käyttäjiä ja tietuemäärä suuri.	Erittäin hidas. Massalisäysmahdollisuus parantaa suorituskykyä.	Soveltuu melko huonosti IoT-datan käsittelyyn.
Elasticsearch	Nopea, koska käyttää päämuistia ja dokumentit pakataan. Nopea kohtalaisella tietuemäärällä.	Nopea pienehköillä tietuemäärillä (100 000).	Käytetään hakukoneena.
MongoDB	Nopea, kun tietuemäärä pieni tai kohtalainen. Suoritusaika alkaa kasvaa tietokannan koon kasvaessa. Nopea, kun käytössä useampi kokoelma.	Kohtalainen, lukua hitaampi. Melko hidas tietuemäärällä 600 000. Optimointi astuu käyttöön tietuemäärällä 700 000. Toiseksi nopein kahdessa eri tutkimuksessa. Nopea, kun käytössä yksi kokoelma (massalisäys).	Isäntä-orja -arkkitehtuuri pullonkaula useilla samanaikaisilla käyttäjillä. Asynkroniset luku- ja kirjoitusoperaatiot. Vanhoissa versioissa hajautustapa hidasti. Hyvä kirjoituspainotteisen sensoridatan käsittelyssä.
OrientDB	Hitaat lukuoperaatiot, koska tietueet luetaan levyltä.	Erittäin hidas - keskiverto.	Tietomalliltaan hybridi, joten suorituskyvyn vertailu muihin hankalaa.
RavenDB	Erittäin hidas.	Erittäin hidas.	Vain yhdessä tutkimuksessa.

Taulukko 17: Dokumenttitietokantojen tulokset

Sarakeperheiden etuna on hyvä horisontaalinen skaalautuvuus. Ne soveltuvat erityisesti erittäin suurten tietomäärien käsittelyyn hajautetuissa, paljon yhtäaikaisia käyttäjiä sisältävissä käyttökohteissa. Tietokantojen optimointiominaisuudet tulevat käyttöön

vasta suuremmilla tietuemäärillä. Sarakeperheistä on eniten tutkittu Cassandran ja HBasen suorituskykyä. Tutkimustulokset sarakeperheiden osalta on esitetty taulukossa 18.

Tietokanta	**Lukupainotteiset kuormitukset** B (95/5), C (100), 80/20	**Päivityspainotteiset kuormitukset** A (50/50), G (5/95), H (100)	**Muita huomioita** hajautus, samanaikaiset käyttäjät, skaalautuvuus
Cassandra	Hyvä suurilla tietuemäärillä. Nopeus kasvaa tietokannan koon kasvaessa. Lukua hidastaa tietueiden kokoaminen luettavaksi levyllä, mikä on huomattavissa erittäin suurilla tietuemäärillä.	Kirjoitusoptimoitu. Hyvä tai erittäin hyvä. Nopeat kirjoitusoperaatiot, koska käyttää välimuistia.	Ei sovellu hyvin välimuistiksi. Erittäin hyvin skaalautuva. Suoritusteho kasvaa lineaarisesti.
HBase	Cassandraa hitaampi. Lukua hidastaa tietueiden osien kerääminen levysijainneista. Uudemmissa versioissa BlockCache, mikä parantaa lukunopeutta.	Nopea tietyissä tapauksissa, mutta Cassandraa hitaampi.	
Hypertable	Hyvä, paljon Cassandraa nopeampi.	Hieman Cassandraa hitaampi.	Vain yhdessä tutkimuksessa.

Taulukko 18: Sarakeperhetietokantojen tulokset

Tietokannan suorituskykyyn vaikuttaa merkittävästi se, käsitelläänkö tietoa muistissa vai levyllä. Tiedon käsittely muistissa sekä välimuistin käyttö lyhentävät suoritusaikaa. Erityisesti päivitysten puskurointi muistiin tekee kirjoitusviiveestä erittäin lyhyen. Levyoperaatioiden suoritus on huomattavasti hitaampaa. Jos tietokannan kirjoitusmekanismi hajauttaa tietueet eri tiedostoihin, se hidastaa lukuoperaatioiden suoritusta. Suorituskykyä parantaa myös levytapahtumien käsittely asynkronisesti. Tietokannan koon vaikutus suorituskykyyn alkaa näkyä vasta suurilla tietuemäärillä, sillä monissa NoSQL-tietokannoissa optimointiominaisuudet astuvat tällöin käyttöön. Suorituskyky siis usein kasvaa tietokannan koon kasvaessa. Myös tietorakenteella on vaikutusta suorituskykyyn. Jos tiedot on sijoitettu yhteen kokoelmaan, tauluun tai hajautukseen, on mahdollista käyttää massalisäystä, mikä nopeuttaa kirjoitusoperaatioiden suoritusta erityisesti suurilla tietuemäärillä yksittäiseen lisäämiseen verrattuna. Toisaalta useampi kokoelma nopeuttaa lukuoperaatioita ja tekee kyselyistä yksinkertaisempia sekä vähentää toisteisuutta, mikä taas pienentää tietokannan kokoa. Replikointimalleista isännätön arkkitehtuuri sallii paremmin suuren määrän samanaikaisia käyttäjiä kuin isäntä-orja -arkkitehtuuri.

9 Johtopäätökset

Tämän tutkimuksen tavoitteena oli kartoittaa erilaisten NoSQL-tietokantojen suorituskykyä aihetta käsittelevien tutkimusten pohjalta. Tutkimuksena tuloksena saatiin tietoa sekä NoSQL-tietomallien että tarkemmalla tasolla niihin kuuluvien tietokantojen suorituskyvystä sekä soveltuvuudesta erilaisiin käyttökohteisiin. Yksinkertaiset testiympäristöt vastaavat kuitenkin harvoin todellisten sovellusten toimintaa. Lisäksi tutkimusten vertailu keskenään on hankalaa, eikä erilaisissa ympäristöissä ja eri työkaluilla tehtyjä tutkimuksia voida suoraan vertailla keskenään. NoSQL-tietokantojen kehitys on tällä hetkellä nopeaa ja uusia tietokantoja ja niiden versioita julkaistaan jatkuvasti. Uusien versioiden myötä tietokantoihin on voinut tulla merkittäviäkin suorituskykyyn vaikuttavia parannuksia, joten tutkimuksen tulokset voivat yksittäisten tuotteiden osalta vanhentua nopeasti. Tutkimuksen tuloksia voidaankin kuitenkin käyttää suuntaa-antavana tietolähteenä suorituskyvyltään parhaan mahdollisen tietomallin ja tietokannan valinnassa. Suorituskyvyn kannalta olennaisinta on määritellä tarkasti tietokannan käyttökohteen vaatimukset. Parhaan mahdollisen suorituskyvyn tavoittelu voi kuitenkin johtaa kompromisseihin muiden tärkeiden ominaisuuksien suhteen. Tässä tutkimuksessa vertailtiin NoSQL-tietokantojen suorituskykyä ja NewSQL-tietokantoja sivuttiin hyvin lyhyesti. Kiinnostava jatkotutkimuksen aihe olisikin NoSQL- ja NewSQL-tietokantojen suorituskyvyn vertailu.

10 Viiteluettelo

Abramova V. and Bernardino J. 2013. NoSQL databases: MongoDB vs Cassandra. In: *Proc. of the International C* Conference on Computer Science and Software Engineering (C3S2E),* 14–22.

Abramova V., Bernardino J. and Furtado P. 2014a. Experimental evaluation of NoSQL databases. *International Journal of Database Management Systems 6*, 1–16.

Abramova V., Bernardino J. and Furtado P. 2014b. Which NoSQL database? A performance overview. *Open Journal of Databases (OJDB) 1 (2),* 17-24.

Abubakar Y., Adeyi T. S. and Gaubo Auta I. 2014. Performance evaluation of NoSQL systems using YCSB in a resource Austere environment. *International Journal of Applied Information Systems 7*, 23–27.

Angles R. and Gutierrez C. 2008. Survey of graph database models. *ACM Computing Surveys 40,* 1–39.

Aqua Studio. 2019. AquaFold, Inc. https://www.aquafold.com/aquadatastudio Viitattu 23.4.2019.

Aslett M. 2011. How will the database incumbents respond to NoSQL and NewSQL? The 451 Group. https://cs.brown.edu/courses/cs227/archives/2012/papers/newsql/aslett-newsql.pdf Viitattu 23.4.2019

Bazar C. and Iosif C. S. 2014. The transition from RDBMS to NoSQL. A comparative analysis of three popular non-relational solutions: Cassandra, MongoDB and Couchbase. *Database Systems Journal 2*, 49 – 59.

Oracle Berkeley DB. 2019. https://www.oracle.com/database/technologies/related/berkeleydb.html viitattu 23.4.2019.

Brewer E. 2012. CAP twelve years later: How the "rules" have changed. *Computer 45*, 23–29.

Cassandra. 2016. The Apache Software Foundation. http://cassandra.apache.org/ Viitattu 3.2.2019

Chamberlin, D.D. and Boyce, R.F. 1974. SEQUEL: A structured english query language. In: *Proc. of the 1974 ACM SIGFIDET Workshop on Data Description, Access and Control*, 249–264.

Chamberlin, D. 2002. XQuery: An XML query language. *IBM Systems Journal 41*, 597–615.

Chang F., Dean J., Ghemawat S., Hsieh W. S., Wallach D.A., Burrows M., Chandra T., Fikes A. and Gruber R. 2008. Bigtable: a distributed storage system for structured data. *ACM Transactions on Computer Systems 26*, 1–26.

Cisco ParStream. 2018. Cisco ParStream Manual. https://www.cisco.com/c/dam/en/us/td/docs/cloud-systems-management/cisco-

edge-fog-fabric/1_6_0/Cisco-ParStream-Manual-6-0-1.pdf?dtid=osscdc000283 Viitattu 24.2.2019

Clark J. and DeRose S. 1999. XML Path Language (XPath). https://www.w3.org/TR/1999/REC-xpath-19991116/ Viitattu 23.4.2019

Cloud Spanner. 2019. Google. https://cloud.google.com/spanner/ Viitattu 23.4.2019

Codd E. F. 1970. A relational model of data for large shared data banks. *Communications of the ACM 13 (6),* 377-378.

Cooper B. F., Silberstein A., Tam E., Ramakrishnan R. and Sears R. 2010. Benchmarking cloud serving systems with YCSB. In: *Proc. of the 1st ACM symposium on Cloud computing,* 143-154.

Couchbase. 2019. Couchbase. https://www.couchbase.com/ Viitattu 3.2.2019

CouchDB. 2019. The Apache Software Foundation. http://couchdb.apache.org/ Viitattu 3.2.2019

Dahlberg T. 2015. Miten hallitsemme digitaalista tietoa vuonna 2040 jos sitä on 33 miljoona kertaa nykyistä enemmän? https://www.sfs.fi/files/8009/Dahlberg-vuo-siseminaari-2015.pdf Viitattu 22.4.2019

Dayarathna, M. and Suzumura, T. 2014. Graph database benchmarking on cloud environments with XGDBench. *Automated Software Engineering 21*, 509–533.

DB-Engines. 2019. DB-Engines Ranking. https://db-engines.com/en/ranking Viitattu 6.4.2019

Dean J. and Ghemawat S. 2008. MapReduce: simplified data processing on large clusters. *Communications of the ACM 51*, 107.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W. 2007. Dynamo: Amazon's highly available key-value store. In: *Proc. of twenty-first ACM SIGOPS symposium on Operating systems principles*, 205-220.

Edlich, S. NOSQL Databases. http://nosql-database.org/ Viitattu 24.2.2019

Elasticsearch. 2019. Elasticsearch B.V. https://www.elastic.co/products/elasticsearch Viitattu 3.2.2019

Elmasri R. and Navathe S. 1989. *Fundamentals of Database Systems.* Benjamin Cummings.

Elmasri R. and Navathe S. 2004. *Fundamentals of Database Systems*. 4. ed. Addison-Wesley.

Fielding, R. T. 2000. *Architectural Styles and the Design of Network-based Software Architectures.* Doctoral dissertation. University of California.

Gandini A., Gribaudo M., Knottenbelt W. J., Osman R. and Piazzolla P. 2014. Performance evaluation of NoSQL databases. In: *Computer Performance Engineering,*

Lecture Notes in Computer Science. European Workshop on Performance Engineering, 16–29.

GraphDB. 2019. Ontotext. Inc. https://www.ontotext.com/products/graphdb/ Viitattu 23.4.2019

Gray J. 1981. The transaction concept: virtues and limitations. In: *Proc. of the 7th International Conference on Very Large Databases*, 144-154.

Grolinger, K., Higashino, W.A., Tiwari, A. and Capretz, M.A., 2013. Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing 2*, 22.

Gudivada V. N., Rao D. and Raghavan V. 2014. NoSQL systems for big data management. In: *2014 IEEE 10th World Congress on Services*, 190-197.

Haerder T. and Reuter A. 1983. Principles of transaction-oriented database recovery. *ACM Computing Surveys 15*, 287–317.

Hashem H. and Ranc D. 2016. Evaluating NoSQL document oriented data model. In: *IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW),* 51-56.

HBase. 2019. https://hbase.apache.org/ Viitattu 23.4.2019

Hecht R. and Jablonski S. 2011. NoSQL evaluation: a use case oriented survey. In: *International Conference on Cloud and Service Computing,* 336–341.

Hilbert M. and Lopez P. 2011. The world's technological capacity to store, communicate, and compute information. *Science 332*, 60–65.

Hypertable. 2014. Hypertable Inc. http://www.hypertable.org/ Viitattu 6.4.2019

Kielitoimiston sanakirja. 2018. Kotimaisten kielten keskus ja Kielikone Oy. https://www.kielitoimistonsanakirja.fi/digitaalinen Viitattu 22.4.2019

Klein J., Gorton I., Ernst N., Donohoe P., Kim Pham and Chrisjan Matser. 2015. Performance evaluation of NoSQL databases: A case study. In: *Proc. of the 1st Workshop on Performance Analysis of Big Data Systems*, 5-10.

Li Y. and Manoharan S. 2013. A performance comparison of SQL and NoSQL databases. In: *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM),* 15-19.

MemSQL. 2019. MemSQL, Inc. https://www.memsql.com/ Viitattu 24.2.2019

MongoDB. 2019. MongoDB Inc. https://www.mongodb.com/ Viitattu 3.2.2019

Nayak A., Poriya A. and Poojary D. 2013. Type of NOSQL databases and its comparison with relational databases. In: *International Journal of Applied Information Systems 5,* 16–19.

Neo4j. 2019. Neo4j, Inc. https://neo4j.com Viitattu 3.2.2019

Oracle NoSQL. 2019. Oracle. https://cloud.oracle.com/nosqldatabase Viitattu 3.2.2019

Oracle. 2019. Oracle. https://www.oracle.com/database/ Viitattu 23.4.2019

OrientDB. 2019. Callidus Software Inc. https://orientdb.com/ Viitattu 3.2.2019

Palovska H. 2015. What can NoSQL serve an enterprise. *Journal of System Integrations 6 (3)*, 44-49.

Phan T.A.M, Nurminen J. K. and Di Francesco M. 2014. Cloud databases for internet-of-Things data. In: *IEEE International Conference on Internet of Things (IThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom)*, 117–124.

PostgreSQL. 2019. PostgreSQL Global Development Group. https://www.postgresql.org/ Viitattu 23.4.2019

Rabl T., Gómez-Villamor S., Sadoghi M., Muntés-Mulero V., Jacobsen H-A and Mankovskii S. 2012. Solving big data challenges for enterprise application performance management. In: *Proc. of the VLDB Endowment 5*, 1724–1735.

RavenDB. 2019. Hibernating Rhinos. https://ravendb.net/ Viitattu 3.2.2019

Redis. 2019 Redislabs. https://redis.io/ Viitattu 3.2.2019

Reinsel D., Gantz J. and Rydning J. 2018. The digitization of the world from edge to core. *IDC White Paper US44413318*.

Riak KV. 2019. Basho. http://basho.com/products/riak-kv/ Viitattu 3.2.2019

Scalaris. 2018. Zuse Institute Berlin http://scalaris.zib.de/ Viitattu 3.2.2019

SQL Server. 2019. Microsoft. https://www.microsoft.com/fi-fi/sql-server/sql-server-2019 Viitattu 23.4.2019

Tang E. and Fan Y. 2016. Performance comparison between five NoSQL databases. In: *7th International Conference on Cloud Computing and Big Data (CCBD),* 105 – 109.

Tarantool. 2019. Tarantool. https://tarantool.io/en/ Viitattu 3.2.2019

Ullman J. D. 1988. *Principles of Database and Knowledgebase Systems*. Computer Science Press.

Voldemort. 2019. https://www.project-voldemort.com/voldemort/ Viitattu 3.2.2019

VoltDB. 2019. VoltDB, Inc. 2019. https://www.voltdb.com/ Viitattu 24.2.2019